K 6

```
EEEEEEEEEEEEEEEE  RRRRRRRRRRRRR     FFFFFFFFFFFFFFFF
EEEEEEEEEEEEEEEE  RRRRRRRRRRRRR     FFFFFFFFFFFFFFFF
EEEEEEEEEEEEEEEE  RRRRRRRRRRRRR     FFFFFFFFFFFFFFFF
EEE               RRR          RRR  FFF
EEE               RRR          RRR  FFF
EEE               RRR          RRR  FFF
EEE               RRR          RRR  FFF
EEE               RRR          RRR  FFF
EEE               RRR          RRR  FFF
EEEEEEEEEEEE      RRRRRRRRRRRRR     FFFFFFFFFFFF
EEEEEEEEEEEE      RRRRRRRRRRRRR     FFFFFFFFFFFF
EEEEEEEEEEEE      RRRRRRRRRRRRR     FFFFFFFFFFFF
EEE               RRR   RRR         FFF
EEE               RRR   RRR         FFF
EEE               RRR   RRR         FFF
EEE               RRR      RRR      FFF
EEE               RRR      RRR      FFF
EEE               RRR      RRR      FFF
EEEEEEEEEEEEEEEE  RRR          RRR  FFF
EEEEEEEEEEEEEEEE  RRR          RRR  FFF
EEEEEEEEEEEEEEEE  RRR          RRR  FFF
```

```
_$

Va
--
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
```

H 2

```
**FILE**ID**MEMORYS

MM      MM EEEEEEEEEE MM      MM   000000   RRRRRRRR   YY      YY   SSSSSSSS
MM      MM EEEEEEEEEE MM      MM   000000   RRRRRRRR   YY      YY   SSSSSSSS
MMMM  MMMM EE         MMMM  MMMM 00      00 RR      RR YY      YY SS
MMMM  MMMM EE         MMMM  MMMM 00      00 RR      RR YY      YY SS
MM  MM  MM EE         MM  MM  MM 00      00 RR      RR   YY  YY   SS
MM  MM  MM EE         MM  MM  MM 00      00 RR      RR   YY  YY   SS
MM      MM EEEEEEEE   MM      MM 00      00 RRRRRRRR       YY       SSSSSS
MM      MM EEEEEEEE   MM      MM 00      00 RRRRRRRR       YY       SSSSSS
MM      MM EE         MM      MM 00      00 RR  RR         YY             SS
MM      MM EE         MM      MM 00      00 RR  RR         YY             SS
MM      MM EE         MM      MM 00      00 RR    RR       YY             SS    ....
MM      MM EE         MM      MM 00      00 RR    RR       YY             SS    ....
MM      MM EEEEEEEEEE MM      MM   000000   RR      RR     YY     SSSSSSSS      ....
MM      MM EEEEEEEEEE MM      MM   000000   RR      RR     YY     SSSSSSSS      ....

LL           IIIIII      SSSSSSSS
LL           IIIIII      SSSSSSSS
LL             II      SS
LL             II      SS
LL             II      SS
LL             II      SS
LL             II        SSSSSS
LL             II        SSSSSS
LL             II              SS
LL             II              SS
LL             II              SS
LL             II              SS
LLLLLLLLLL   IIIIII    SSSSSSSS
LLLLLLLLLL   IIIIII    SSSSSSSS
```

000

```
C
C Version:      'V04-000'
C
C****************************************************************************
C*                                                                          *
C*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
C*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
C*  ALL RIGHTS RESERVED.                                                    *
C*                                                                          *
C*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
C*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
C*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
C*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
C*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
C*  TRANSFERRED.                                                            *
C*                                                                          *
C*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
C*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
C*  CORPORATION.                                                            *
C*                                                                          *
C*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
C*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
C*                                                                          *
C*                                                                          *
C****************************************************************************
C
C
C       Author Brian Porter                     Creation Date 30-DEC-1980
C
C       Modified by:
C
C       V03-012 SAR0258         Sharon A. Reynolds      25-Apr-1984
C               - Fixed a problem in the micro vax memory support.
C               - Added an sye update that adds decoding for
C               MA780 pcsr.
C
C       V03-011 SAR0246         Sharon A. Reynolds,     9-Apr-1984
C               Added micro vax memory support.
C
C       V03-010 SAR0202         Sharon A. Reynolds,     27-Feb-1984
C               Added the MEMORY_REGISTER_UV1 routine. Because memory
C               support for the micro vax is not ready it will output a
C               message. This was done so that the link of ERF would not
C               fail.
C
C       V03-009 SAR0178         Sharon A. Reynolds,     30-Nov-1983
C               Fixed the entry headers for 11/750 and 11/730 memories.
C
C       V03-008 SAR0169         Sharon A. Reynolds,     28-Oct-1983
C               Added an SYE update that:
C               - Modified MS780E reporting to weight array numbers
C                 of the second cotnroller by 8.
C               - Also added code to count multiple errors being
C                 logged by one controller.
C
C       V03-007 SAR0083         Sharon A. Reynolds,     20-Jun-1983
C               Changed the carriage control in the 'format' statements
```

```
C               for use with ERF.
C
C       V03-006 SAR0058         Sharon A. Reynolds,     2-Jun-1983
C               Made 'memory_display' a subroutine and added a routine
C               that gets the necessary queue information so that 'memory_q'
C               and 'memory_display' can be linked with separate images.
C               Removed brief and cryptic code.
C
C       v03-005 BP0005          Brian Porter,           14-APR-1983
C               11/730 syndrome bits are inverted so...
C
C       v03-004 BP0004          Brian Porter,           08-MAR-1983
C               Corrected 11/730 memory size.
C
C       v03-003 BP0003          Brian Porter,           09-NOV-1982
C               Corrected bank value for m8750 memories.
C
C       v03-002 BP0002          Brian Porter,           13-AUG-1982
C               Corrected ms780e memory start address.
C
C       v03-001 BP0001          Brian Porter,           04-APR-1982
C               Corrected output conversion error.
C**

        Subroutine MEMORY (lun,option)

C++
C       Functional description:
C
C       This module extracts the varies arguments from the memory error
C       entry and calls memory_q.  The format of a memory error entry is as
C       follows.  The area occupied by the adaptor TR and registers is
C       repeated for the 'number of controllers' times.  The register area
C       size is dependent on the memory controller of the system being logged.
C       The error pc and error psl are manufactured by VMS and have no meaning,
C       therefore they are not printed in any of the reports.
C
C       +-----------------------------+
C       :                             :
C       +--                         --+
C       :        header space         :
C       +--                         --+
C       :                             :
C       +--                         --+
C       :                             :
C       +-----------------------------+
C       :    number of controllers    :
C       +-----------------------------+
C       : adprtor tr# (or equivalent) :
C       +-----------------------------+
C       :                             :
C       +--                         --+
C       :                             :
C       +--     memory registers    --+
C       :                             :
C       +--                         --+
C       :                             :
```

```
C          +------------------------------+
C          |           error pc           |
C          +------------------------------+
C          |           error psl          |
C          +------------------------------+
C--



      include 'src$:msghdr.for /nolist'
      include 'src$:syecom.for /nolist'

      byte              lun

      character*1       option

      integer*4         buffer(0:120)
      integer*4         controller_count
      Integer*4         I
      Integer*4         lib$extzv
      integer*4         compress4
      integer*4         error_array
      integer*4         error_bank
      integer*4         error_bit
      integer*4         page_count
      integer*4         array_code
      integer*4         decode_ecc

      equivalence       (emb(16),buffer)
      equivalence       (buffer(0),controller_count)

      logical*1         l0011
      logical*1         l0016

      if (
     1 lib$extzv(24,8,emb$l_hd_sid) .eq. 255
     1 .or.
     1 lib$extzv(24,8,emb$l_hd_sid) .eq. 1
     1 ) then

      j = 0

      do 20,i = 1,controller_count

      if (option .ne. 'R'
     1 .and.
     1 j .eq. 0) then

      call header (lun)

      if (emb$w_hd_entry .eq. 5) then

      call logger (lun,'SBI ALERT')
```

```
        else if (emb$w_hd_entry .eq. 8) then

        call logger (lun,'FATAL MEMORY ERROR')
        else

        call logger (lun,'MEMORY ERROR')
        endif
        endif


C
C       MS780C
C
        if (lib$extzv(5,3,buffer(2 + j)) .eq. 0) then

        if (emb$w_hd_entry .ne. '05'x) then

        if (lib$extzv(28,1,buffer(4 + j)) .eq. 1) then

        if (lib$extzv(4,1,buffer(2 + j)) .eq. 0) then

        call memory_g (emb$l_hd_sid,buffer(1 + j),
        1 lib$extzv(24,4,buffer(4 + j)),lib$extzv(21,1,buffer(4+j)),
        1 decode_ecc(lib$extzv(0,8,buffer(4 + j)),buffer(2 + j)))

        else if (lib$extzv(4,1,buffer(2 + j)) .eq. 1) then

        call memory_g (emb$l_hd_sid,buffer(1 + j),
        1 lib$extzv(24,4,buffer(4 + j)),lib$extzv(23,1,buffer(4+j)),
        1 decode_ecc(lib$extzv(0,8,buffer(4 + j)),buffer(2 + j)))
        endif
        endif
        endif

C
C Full report output the TR# and call the MS780C routine to
C decode/output the remainder of the entry.
C
        if (option .eq. 'S') then

        call linchk (lun,2)

        write(lun,10) buffer(1 + j)
10      format(/' ','CONTROLLER AT TR #',i<compress4 (buffer(1 + j))>,'.')

        call ms780c (lun,buffer(2 + j))
        endif


        j = j + 4


C
C       MS780E
C
        else if (lib$extzv(5,3,buffer(2 + j)) .eq. 3) then

        if (emb$w_hd_entry .ne. '05'x) then

        if (lib$extzv(28,1,buffer(4 + j)) .eq. 1) then

        call memory_q (emb$l_hd_sid,buffer(1 + j),
        1 lib$extzv(24,3,buffer(4 + j)),lib$extzv(22,2,buffer(4+j)),
        1 decode_ecc(lib$extzv(0,7,buffer(4 + j)),buffer(2 + j)))
        Endif

        If (lib$extzv(28,1,buffer(5 + j)) .eq. 1) then

        call memory_q (emb$l_hd_sid,buffer(1 + j),
        1 (lib$extzv(24,3,buffer(5 + j))+8),lib$extzv(22,2,buffer(5+j)),
        1 decode_ecc(lib$extzv(0,7,buffer(5 + j)),buffer(2 + j)))
        Endif

        If (
        1 lib$extzv(18,2,buffer(2 + j)) .ne. 0
        1 .or.
        1 lib$extzv(7,1,buffer(3 + j)) .eq. 1
        1 .or.
        1 lib$extzv(7,1,buffer(4 + j)) .eq. 1
        1 .or.
        1 lib$extzv(7,1,buffer(5 + j)) .eq. 1
        1 ) then

        call memory_q (emb$l_hd_sid,buffer(1 + j),-1,-1,-1)
        endif
        endif

C
C Full report output the TR# and call the MS780E routine to
C decode/output the remainder of the entry.
C
        if (option .eq. 'S') then

        call linchk (lun,2)
        write(lun,10) buffer(1 + j)

        call ms780e (lun,buffer(2 + j))
        endif


        j = j + 5

C
C       MA780
C
        else if (lib$extzv(5,3,buffer(2 + j)) .eq. 2) then

        if (emb$w_hd_entry .ne. '05'x) then

        if (lib$extzv (28,1,buffer(6 + j)) .eq. 1) then

        call memory_q (emb$l_hd_sid,buffer(1 + j),
        1 lib$extzv(24,4,buffer(6 + j)),lib$extzv(23,1,buffer(6+j)),
        1 decode_ecc(lib$extzv(0,8,buffer(6 + j)),buffer(2 + j)))
        Endif

        If (
        1 lib$extzv(26,6,buffer(3 + j)) .ne. 0
        1 .or.
        1 lib$extzv(14,2,buffer(4 + j)) .ne. 0
        1 .or.
        1 lib$extzv(28,1,buffer(4 + j)) .eq. 1
        1 .or.
        1 lib$extzv(30,2,buffer(4 + j)) .ne. 0
        1 .or.
        1 lib$extzv(31,1,buffer(6 + j)) .eq. 1
        1 ) then

        call memory_q (emb$l_hd_sid,buffer(1 + j),-1,-1,-1)
        endif
        endif

C
C Full report output the TR# and call the MA780 routine to
C decode/output the remainder of the entry.
C
        if (option .eq. 'S') then

        call linchk (lun,2)
        write(lun,10) buffer(1 + j)

        call ma780 (lun,buffer(2 + j))
        endif

        j = j + 9
        endif

20      continue


C
C       11/750
C
        else if (lib$extzv(24,8,emb$l_hd_sid) .eq. 2) then

        If (option .eq. 'S'
        1 .OR.
        2  option .eq. 'R') then

        error_array = lib$extzv(9,15,buffer(2))

        l0011 = .false.

        l0016 = .false.

        if (jiand(buffer(4),'01000000'x) .ne. 0) then

        l0016 = .true.
        else

        l0011 = .true.
        endif

        do 30,i = 0,15,2

        array_code = lib$extzv(i,2,buffer(4))

        if (l0016) then

        if (array_code .eq. 3) then

        error_array = error_array - 512

        if (lib$extzv(17,1,buffer(2)) .eq. 0) then

        error_bank = 0
        else

        error_bank = 1
        endif

        else if (array_code .eq. 2) then

        error_array = error_array - 2048

        error_bank = lib$extzv(19,2,buffer(2))
        endif

        else if (l0011) then

        if (array_code .eq. 1) then

        error_array = error_array - 256

        error_bank = 0

        else if (array_code .eq. 3) then

        error_array = error_array - 512

        if (lib$extzv(17,1,buffer(2)) .eq. 0) then

        error_bank = 0
        else

        error_bank = 1
        endif
        endif
        endif

        if (error_array .le. 0) then

        error_array = i/2

        goto 40
        endif

30      continue
```

```

40      if (lib$extzv(29,1,buffer(2)) .eq. 1) then

        call memory_q (emb$l_hd_sid,lib$extzv(0,3,buffer(1)),
        1 error_array,error_bank,decode_ecc(lib$extzv(0,7,buffer(2))))

        else if (lib$extzv(31,1,buffer(2)) .eq. 1) then

        call memory_q (emb$l_hd_sid,lib$extzv(0,3,buffer(1)),
        1 error_array,error_bank,-1)
        endif
        endif

        If (option .NE. 'R') then

        call header (lun)

        If (emb$w_hd_entry .eq. 8) then

        call logger (lun,'FATAL MEMORY ERROR')
        else

        call logger (lun,'MEMORY ERROR')
        endif
        endif

        if (option .eq. 'S') then

        call linchk (lun,2)
        write(lun,50) lib$extzv(0,3,buffer(1))
50      format(/' ','CONTROLLER AT SLOT INDEX #',
        1 i<compress4 (lib$extzv(0,3,buffer(1)))>,'.')

        call ms750 (lun,buffer(2))
        endif

C
C       11/730
C
        else if (lib$extzv(24,8,emb$l_hd_sid) .eq. 3) then

        If (option .eq. 'S'
        1 .or.
        2 option .eq. 'R') then
c
c       11/730 syndrome bits are inverted so...
c

        error_bit = decode_ecc (lib$extzv(0,7,not(lib$extzv(0,7,buffer(0)))))

        error_array = lib$extzv(9,15,buffer(0))

        if (lib$extzv(24,1,buffer(2)) .eq. 1) then

        page_count = 1024
        else

```

```
                page_count = 256
                endif

                do 60,i = 0,15

                if (lib$extzv(i,1,buffer(2)) .eq. 1) then

                error_array = error_array - page_count

                if (error_array .le. 0) then

                error_array = i/2

                goto 65
                endif
                endif

        60      continue

        65      if (page_count .eq. 1024) then

                error_bank = lib$extzv (19,2,buffer(0))
                else

                error_bank = lib$extzv (17,1,buffer(0))
                endif

                if (error_bit .eq. -1) then

                call memory_q (emb$l_hd_sid,0,
                1 error_array,error_bank,-1)
                else

                call memory_q (emb$l_hd_sid,0,
                1 error_array,error_bank,error_bit)
                endif
                endif

                If (option .ne. 'R') then

                call header (lun)

                If (emb$w_hd_entry .eq. 8) then

                call logger (lun,'FATAL MEMORY ERROR')
                else

                call logger (lun,'MEMORY ERROR')
                endif

                call ms730 (lun,buffer)
                endif


        c       UVAX1
        c
```

```
MA7E

PROG

 0
 1
 2

ENTR

 0-

VARI

 2-
 2-
 2-
 2-
 2-
 2-
 2-
 2-
 2-
AP-
AP-
 2-
 2-
 2-
 2-
 2-
```

```
        else if (lib$extzv(24,8,emb$l_hd_sid) .eq. 7) then

        do 80,i = 1,16

        if (lib$extzv(15,1,buffer(i)) .eq. 1) then

        call memory_q (emb$l_hd_sid,0,i,-1,-1)
        endif

 80     continue

        if (option .ne. 'C'
        1 .and.
        1 option .ne. 'R') then

        call header (lun)

        call logger (lun,'FATAL MEMORY ERROR')
        endif

        if (option .eq. 'S') then

        call memory_register_uv1 (lun,buffer(0))
        endif

c
c       The IF-THEN-ELSE should be expanded at this point to add
c       additional CPU memory support.
c
        endif

        return
        end
```

```
                                                                   F  3
MEMORY                                                            16-Sep-1984 00:07:33     VAX-11 FORTRAN V3.4-56                Page  11
                                                                   5-Sep-1984 14:01:18     DISK$VMSMASTER:[ERF.SRC]MEMORYS.FOR;1

PROGRAM SECTIONS

    Name                                   Bytes   Attributes

  0 $CODE                                   2633   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                   218   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                   928   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG
  3 EMB                                      512   PIC OVR REL GBL   SHR NOEXE   RD   WRT LONG
  4 SYECOM                                    44   PIC OVR REL GBL   SHR NOEXE   RD   WRT LONG

    Total Space Allocated                   4335


ENTRY POINTS

    Address   Type  Name

  0-00000000        MEMORY


VARIABLES

    Address   Type  Name                                   Address   Type  Name

  2-00000018  I*4   ARRAY_CODE                           3-00000010  I*4   CONTROLLER_COUNT
  4-00000012  L*1   CP_11750                             4-00000011  L*1   CP_11780
  4-00000013  L*1   CP_117ZZ                             4-00000014  L*4   CRYPTK_FLAG
  4-0000000D  I*4   DEV_CHAR                             3-00000000  I*4   EMB$L_HD_SID
  3-00000004  I*2   EMB$W_HD_ENTRY                       3-0000000E  I*2   EMB$W_HD_ERRSEQ
  4-0000001E  L*1   END_VALUE                            4-0000001D  L*1   EOF_FLAG
  2-00000008  I*4   ERROR_ARRAY                          2-0000000C  I*4   ERROR_BANK
  2-00000010  I*4   ERROR_BIT                            4-00000004  L*4   FORMS
  2-00000004  I*4   I                                    2-0000001C  I*4   J
  2-00000000  L*1   L0011                                2-00000001  L*1   L0016
  4-0000000C  L*1   LINES                                4-00000027  I*4   LSTLUN
 AP-00000004@ L*1   LUN                                  4-0000001F  I*4   MAILBOX_CHANNEL
 AP-00000008@ CHAR  OPTION                               4-0000002B  CHAR  OPTIONS
  2-00000014  I*4   PAGE_COUNT                           4-00000008  L*4   PRINTER
  4-00000000  I*4   RECCNT                               4-00000023  I*4   RECORD_SIZE
  4-00000019  L*1   VALID_CLASS                          4-0000001A  L*1   VALID_CPU
  4-0000001B  L*1   VALID_ENTRY                          4-0000001C  L*1   VALID_TYPE
  4-00000018  L*1   VOLUME_OUTPUT


ARRAYS

    Address   Type  Name                      Bytes  Dimensions

  3-00000010  I*4   BUFFER                      484  (0:120)
  3-00000000  L*1   EMB                         512  (0:511)
  3-00000006  I*4   EMB$Q_HD_TIME                 8  (2)
```

```
                                                                  G  3
MEMORY                                                      16-Sep-1984 00:07:33    VAX-11 FORTRAN V3.4-56                Page  12
                                                             5-Sep-1984 14:01:18    DISK$VMSMASTER:[ERF.SRC]MEMORYS.FOR;1

LABELS

    Address    Label        Address    Label        Address    Label        Address    Label        Address    Label        Address    Label

  1-0000008E   10'            **       20             **       30        0-0000075C   40        1-000000B0   50'            **       60
  0-000008FC   65             **       80


FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name                                     Type  Name                                     Type  Name

   I*4  COMPRESS4                                 I*4  DECODE_ECC                                     HEADER
   I*4  LIB$EXTZV                                      LINCHK                                         LOGGER
        MA780                                          MEMORY_Q                                       MEMORY_REGISTER_UV1
        MS730                                          MS750                                          MS780C
        MS780E
```

```
        Subroutine MA780 (lun,memory_registers)

c++
c       This routine displays the ma780 memory registers.
c       The format of the multi-port memory sub packet is as follows.
c
c       +-----------------------------------+
c       !   port configuration register     !
c       +-----------------------------------+
c       ! port interface control register   !
c       +-----------------------------------+
c       ! port controller status register   !
c       +-----------------------------------+
c       ! port invalidation control reg     !
c       +-----------------------------------+
c       !       array error register        !
c       +-----------------------------------+
c       ! configuration status register 0   !
c       +-----------------------------------+
c       ! configuration status register 1   !
c       +-----------------------------------+
c       !   maintenance control register    !
c       +-----------------------------------+
c
c--

        Implicit        none
        byte            lun

        integer*4       memory_registers
        integer*4       buffer(8)
        integer*4       port_configuration_register
        integer*4       port_interface_control_register
        integer*4       port_controller_status_register
        integer*4       port_invalidation_control_reg
        integer*4       array_error_register
        integer*4       configuration_status_register0
        integer*4       configuration_status_register1
        integer*4       maintenance_control_register

        equivalence     (buffer(1),port_configuration_register)
        equivalence     (buffer(2),port_interface_control_register)
        equivalence     (buffer(3),port_controller_status_register)
        equivalence     (buffer(4),port_invalidation_control_reg)
        equivalence     (buffer(5),array_error_register)
        equivalence     (buffer(6),configuration_status_register0)
        equivalence     (buffer(7),configuration_status_register1)
        equivalence     (buffer(8),maintenance_control_register)

        integer*4       ram_count
        integer*4       array_count
        integer*4       starting_address
        integer*4       error_syndrome
        integer*4       error_bit
        integer*4       error_array
        integer*4       error_bank
        integer*4       array_init_status_bits
        integer*4       port_type_bits

        equivalence     (ram_count,array_count,starting_address,error_syndrome,
        1 error_bit,error_array,array_init_status_bits,port_type_bits)

        integer*4       compress4
        integer*4       compressc
        integer*4       i
        integer*4       j
        integer*4       lib$extzv
        integer*4       lib$locc
        integer*4       decode_ecc
        integer*4       adapter_tr

        logical*1       diagnostic_mode

        character*32    v1register2(0:1)
        data            v1register2(0)  /'MASTER INTERRUPT ENABLE*'/
        data            v1register2(1)  /'PORT INTERFACE INTERRUPT ENABLE*'/

        character*30    v3register2(23:31)
        data            v3register2(23) /'MARK INTERLOCK IN PROGRESS*'/
        data            v3register2(24) /'MARK TIMEOUT*'/
        data            v3register2(25) /'MARK REQUESTER*'/
        data            v3register2(26) /'OUTPUT BUFFER OVERFLOW*'/
        data            v3register2(27) /'INVALIDATION ACK NOT RECEIVED*'/
        data            v3register2(28) /'OUTPUT BUFFER PARITY ERROR*'/
        data            v3register2(29) /'INVALIDATE LOST ON BDI*'/
        data            v3register2(30) /'BDI PARITY FAULT ON OUTPUT*'/
        data            v3register2(31) /'BDI PARITY FAULT ON INPUT*'/

        character*23    v1register3(1)
        data            v1register3(1)  /'ERROR INTERRUPT ENABLE*'/

        character*25    v2register3(6:8)
        data            v2register3(6)  /'SELF INVALIDATE ENABLE*'/
        data            v2register3(7)  /'INVALIDATION DISABLE*'/
        data            v2register3(8)  /'INHIBIT ADMI ARBITRATION*'/

        character*25    v3register3(10:15)
        data            v3register3(10) /'INTERLOCK GRANT ACCEPTED*'/
        data            v3register3(11) /'INTERLOCK FLIP-FLOP*'/
        data            v3register3(12) /'ARRAY INIT IN PROGRESS*'/
c
c       Cell 13 of this array unused
c
        data            v3register3(14) /'INVALIDATE DATA LOST*'/
        data            v3register3(15) /'INTERLOCK TIMEOUT*'/

        character*30    v4register3(21)
        data            v4register3(21) /'NO C/A ON ADMI WHEN REQUESTED*'/

        character*23    v5register3(22:25,0:1)
        data            v5register3(22,0)/'32-BIT OPERATION*'/
        data            v5register3(22,1)/'64-BIT OPERATION*'/
        data            v5register3(23,0)/'I/O SELECT*'/
        data            v5register3(23,1)/'ARRAY SELECT*'/
        data            v5register3(24,0)/'REQUESTER HAS NO CACHE*'/
        data            v5register3(24,1)/'REQUESTER HAS CACHE*'/
        data            v5register3(25,0)/'ADMI READ*'/
        data            v5register3(25,1)/'ADMI WRITE*'/

        character*32    v6register3(28:31)
        data            v6register3(28) /'MULTIPLE ADMI GRANT*'/
        data            v6register3(29) /'PORT TRANSMITTING DURING FAULT*'/
        data            v6register3(30) /'ADMI MULTIPLE TRANSMITTER FAULT*'/
        data            v6register3(31) /'ADMI COMMAND ABORT*'/

        character*14    v1register4(31:31)
        data            v1register4(31) /'CACHED FORCED*'/

        character*18    v1register5(28:31)
        data            v1register5(28) /'ERROR LOG REQUEST*'/
        data            v1register5(29) /'HIGH ERROR RATE*'/
        data            v1register5(30) /'CRD TAG*'/
        data            v1register5(31) /'MAP PARITY ERROR*'/

        character*27    v1register6(0:1)
        data            v1register6(0)  /'NONCONTIGUOUS ARRAY ERROR*'/
        data            v1register6(1)  /'4K CHIP ARRAY ERROR*'/

        character*28    v1register7(10:11)
        data            v1register7(10) /'MULTIPLE INTERLOCK ACCEPTED*'/
        data            v1register7(11) /'INVALIDATION MAP PRESENT*'/

        character*27    array_init_status
        character*22    port_type


        call movc3 (%val(32),memory_registers,buffer)

        diagnostic_mode = .false.

        if (lib$extzv(4,3,port_interface_control_register) .ne. 0
        1 .or.
        1 lib$extzv(4,2,port_controller_status_register) .ne. 0
        1 .or.
        1 lib$extzv(13,1,port_controller_status_register) .ne. 0
        1 .or.
        1 lib$extzv(8,2,configuration_status_register1) .ne. 0
        1 .or.
        1 lib$extzv(8,6,maintenance_control_register) .ne. 0) then

        diagnostic_mode = .true.

        call linchk (lun,1)

        write(lun,5) port_configuration_register
5       format(/' ',t8,'PRTCFNG',t24,z8.8)
        endif

        call ma780_rega (lun,port_configuration_register)

        call linchk (lun,1)

        write(lun,10) port_interface_control_register
10      format(' ',t8,'PRTCR',t24,z8.8)

        if (.not. diagnostic_mode) then

        call output (lun,port_interface_control_register,v1register2,0,0,1,'0')

        call linchk (lun,1)

        ram_count =  lib$extzv(16,4,port_interface_control_register)

        write(lun,15) ram_count
15      format(' ',t40,'RAM COUNT ',i<compress4 (ram_count)>,'.')

        call output (lun,port_interface_control_register,v3register2,23,23,31,
        1 '0')
        endif

        if (lib$extzv(4,3,port_interface_control_register) .ne. 0) then

        call linchk (lun,1)

        write(lun,17)
17      format(' ',t40,'DIAGNOSTIC MODE')
        endif

        call linchk (lun,1)

        write(lun,20) port_controller_status_register
20      format(' ',t8,'PCSR',t24,z8.8)

        if (lib$extzv(13,1,port_controller_status_register) .ne. 0
        1 .or.
        1 lib$extzv (4,2,port_controller_status_register) .ne. 0) then

        call linchk (lun,1)

        write(lun,17)
        endif

        if (.not. diagnostic_mode) then

        call output (lun,port_controller_status_register,v1register3,1,1,1,'0')

        call output (lun,port_controller_status_register,v2register3,6,6,8,'0')

        call output (lun,port_controller_status_register,v3register3,10,10,12,
        1 '0')

        call output (lun,port_controller_status_register,v3register3,10,14,15,
        1 '0')

        if (jiand(port_controller_status_register,'d0000000'x) .ne. 0) then

        call output (lun,port_controller_status_register,v4register3,21,21,21,
        1 '0')

        call output (lun,port_controller_status_register,v5register3,22,22,25,
        1 '2')

        I = LIB$EXTZV (26,2,port_controller_status_register)

        Call LINCHK (lun,1)
        Write (lun,25) I
25      Format (' ',T40,'"ADMI" PORT #',I<COMPRESS4 (I)>,'.')

        call output (lun,port_controller_status_register,v6register3,28,28,31,
        1 '0')
        endif
        endif

        call linchk (lun,1)

        write(lun,30) port_invalidation_control_reg
30      format(' ',t8,'IVDTCR',t24,z8.8)

        if (.not. diagnostic_mode) then

        do 50,i = 0,15

        if (lib$extzv(i,1,port_invalidation_control_reg) .eq. 1) then

        call linchk (lun,1)

        write(lun,40) i
40      format(' ',t40,'INVALIDATE CACHE DEVICE ID = ',i<compress4 (i)>,'.')
        endif

50      continue

        call linchk (lun,1)

        array_count = lib$extzv(16,3,port_invalidation_control_reg)

        if (array_count .eq. 0
        1 .and.
        1 lib$extzv(0,2,configuration_status_register0) .ne. 0) then

        write(lun,60) 'INVALID ARRAY CONFIGURATION'
60      format(' ',t40,a)
        else

        write(lun,70) array_count + 1
70      format(' ',t40,i<compress4 (array_count + 1)>,
        1 '. ARRAY BOARD(S) PRESENT')
        endif

        starting_address = lib$extzv(20,11,port_invalidation_control_reg)*256

        call linchk (lun,1)

        write(lun,80) starting_address
80      format(' ',t40,'MEMORY BASE ADDRESS = ',
        1 i<compress4 (starting_address)>,'.K')
```

```
                call output (lun,port_invalidation_control_reg,v1register4,31,31,31,
                1 '0')
                endif

                call linchk (lun,1)

                write(lun,90) array_error_register
       90       format(' ',t8,'AER',t24,z8.8)

                if (.not. diagnostic_mode) then

                if (lib$extzv (28,1,array_error_register) .eq. 1) then

                error_syndrome = lib$extzv(0,8,array_error_register)

                call linchk (lun,1)

                write(lun,100) error_syndrome
       100      format(' ',t40,'ERROR SYNDROME = ',z2.2)

                error_bit = decode_ecc (error_syndrome,port_configuration_register)

                call linchk (lun,1)

                if (error_bit .eq. -1) then

                write(lun,110) 'RDS ERROR'
       110      format(' ',t40,a)
                else

                write(lun,120) 'CRD ERROR, CORRECTED BIT #',error_bit,'.'
       120      format(' ',t40,a,i<compress4 (error_bit)>,a)
                endif

                error_array = lib$extzv (24,4,array_error_register)

                call linchk (lun,1)

                write(lun,140) error_array
       140      format(' ',t40,'ARRAY #',i<compress4 (error_array)>,'. IN ERROR')
                endif

                Error_bank = LIB$EXTZV(23,1,array_error_register)

                Call LINCHK (lun,1)
                Write (lun,145) error_bank
       145      Format(' ',T40,
                1 'ARRAY BANK #',I<COMPRESS4 (error_bank)>,'. IN ERROR')

                call output (lun,array_error_register,v1register5,28,28,31,'0')
                endif

                call linchk (lun,1)

                write(lun,170) configuration_status_register0
       170      format(' ',t8,'CSR0',t24,z8.8)
```

```

                if (.not. diagnostic_mode) then

                call output (lun,configuration_status_register0,v1register6,0,0,1,'0')

                array_init_status_bits = lib$extzv(2,2,configuration_status_register0)

                if (array_init_status_bits .eq. 0) then

                array_init_status = 'INITIALIZATION IN PROGRESS*'

                else if (array_init_status_bits .eq. 2) then

                array_init_status = 'MEMORY CONTAINS VALID DATA*'

                else if (array_init_status_bits .eq. 3) then

                array_init_status = 'INITIALIZATION COMPLETE*'
                endif

                call linchk (lun,1)

                write(lun,180) array_init_status(:(lib$locc('*',array_init_status)-1))
        180     format(' ',t40,a)

                do 200,i = 4,7

                if (lib$extzv(i,1,configuration_status_register0) .eq. 1) then

                call linchk (lun,1)

                write(lun,190) 'PORT #',i - 4,'. POWERED DOWN'
        190     format(' ',t40,a,i<compress4 (i - 4)>,a)
                endif

        200     continue

                do 220,i = 8,11

                if (lib$extzv(i,1,configuration_status_register0) .eq. 1) then

                call linchk (lun,1)

                write(lun,210) 'ERROR INTERRUPT FROM PORT #',i - 8,'.'
        210     format(' ',t40,a,i<compress4 (i - 8)>,a)
                endif

        220     continue

                do 240,i = 12,15

                if (lib$extzv(i,1,configuration_status_register0) .eq. 1) then

                call linchk (lun,1)

                write(lun,230) 'PORT #',i - 12,'. OFFLINE'
        230     format(' ',t40,a,i<compress4 (i - 12)>,a)
        endif

240     continue
        endif

        call linchk (lun,1)

        write(lun,250) configuration_status_register1
250     format(' ',t8,'CSR1',t24,z8.8)

        if (.not. diagnostic_mode) then

        do 270,i = 0,6,2

        port_type_bits = lib$extzv(i,2,configuration_scatus_register1)

        if (port_type_bits .eq. 0) then

        port_type = '. NOT PRESENT*'

        else if (port_type_bits .eq. 2) then

        port_type = '. CONNECTED TO AN SBI*'
        endif

        if (port_type_bits .ne. 1
        1 .or.
        1 port_type_bits .ne. 3) then

        call linchk (lun,1)

        write(lun,260) i/2,port_type
260     format(' ',t40,'PORT #',i<compress4 (i/2)>,a<compressc (port_type)>)
        endif

270     continue

        call output (lun,configuration_status_register1,v1register7,10,10,11,
        1 '0')

        do 290,i = 12,15

        if (lib$extzv(i,1,configuration_status_register1) .eq. 1) then

        call linchk (lun,1)

        write(lun,280) 'PORT #',i - 12,'. INVALIDATION ACK RECEIVED'
280     format(' ',t40,a,i<compress4 (i - 12)>,a)
        endif

290     continue
        endif

        if (lib$extzv(8,2,configuration_status_register1) .ne. 0) then

        call linchk (lun,1)

```

```
                write(lun,17)
                endif

                call linchk (lun,1)

                write(lun,295) maintenance_control_register
        295     format(' ',t8,'MAT',t24,z8.8)

                if (lib$extzv(8,6,maintenance_control_register) .ne. 0) then

                call linchk (lun,1)

                write(lun,17)
                endif

                return
                end


PROGRAM SECTIONS

    Name                                         Bytes   Attributes

  0 $CODE                                         2631   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                         919   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                        3188   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

    Total Space Allocated                         6738


ENTRY POINTS

    Address   Type  Name

  0-00000000        MA780


VARIABLES

    Address   Type  Name                                   Address   Type  Name

  2-0000071C  I*4   ADAPTER_TR                           2-00000000  I*4   ARRAY_COUNT
  2-00000014  I*4   ARRAY_ERROR_REGISTER                 2-000006DD  CHAR  ARRAY_INIT_STATUS
  2-00000000  I*4   ARRAY_INIT_STATUS_BITS               2-00000018  I*4   CONFIGURATION_STATUS_REGISTER0
  2-0000001C  I*4   CONFIGURATION_STATUS_REGISTER1       2-000006DC  L*1   DIAGNOSTIC_MODE
  2-00000000  I*4   ERROR_ARRAY                          2-00000710  I*4   ERROR_BANK
  2-00000000  I*4   ERROR_BIT                            2-00000000  I*4   ERROR_SYNDROME
  2-00000714  I*4   I                                    2-00000718  I*4   J
 AP-00000004@ L*1   LUN                                  2-00000020  I*4   MAINTENANCE_CONTROL_REGISTER
 AP-00000008@ I*4   MEMORY_REGISTERS                     2-00000004  I*4   PORT_CONFIGURATION_REGISTER
  2-0000000C  I*4   PORT_CONTROLLER_STATUS_REGISTER      2-00000008  I*4   PORT_INTERFACE_CONTROL_REGISTER
  2-00000010  I*4   PORT_INVALIDATION_CONTROL_REG        2-000006F8  CHAR  PORT_TYPE
  2-00000000  I*4   PORT_TYPE_BITS                       2-00000000  I*4   RAM_COUNT
  2-00000000  I*4   STARTING_ADDRESS
```

```
                                                         E  4
MA780                                                   16-Sep-1984 00:07:33    VAX-11 FORTRAN V3.4-56             Page  23
                                                         5-Sep-1984 14:01:18    DISK$VMSMASTER:[ERF.SRC]MEMORYS.FOR;1

ARRAYS

    Address   Type  Name                   Bytes  Dimensions

  2-00000004  I*4   BUFFER                    32  (8)
  2-00000024  CHAR  V1REGISTER2               64  (0:1)
  2-00000172  CHAR  V1REGISTER3               23  (1)
  2-00000618  CHAR  V1REGISTER4               14  (31:31)
  2-00000626  CHAR  V1REGISTER5               72  (28:31)
  2-0000066E  CHAR  V1REGISTER6               54  (0:1)
  2-000006A4  CHAR  V1REGISTER7               56  (10:11)
  2-00000189  CHAR  V2REGISTER3               75  (6:8)
  2-00000064  CHAR  V3REGISTER2              270  (23:31)
  2-000001D4  CHAR  V3REGISTER3              150  (10:15)
  2-0000026A  CHAR  V4REGISTER3              630  (21)
  2-000004E0  CHAR  V5REGISTER3              184  (22:25, 0:1)
  2-00000598  CHAR  V6REGISTER3              128  (28:31)

LABELS

    Address   Label       Address   Label       Address   Label       Address   Label       Address   Label       Address   Label

  1-000000F4  5'       1-00000109  10'      1-0000011B  15'      1-00000136  17'      1-0000014D  20'      1-0000015E  25'
  1-0000017C  30'      1-0000018F  40'          **      50       1-000001BD  60'      1-000001C4  70'      1-000001EA  80'
  1-00000212  90'      1-00000222  100'     1-0000023E  110'     1-00000245  120'     1-00000253  140'     1-00000274  145'
  1-0000029A  170'     1-000002AB  180'     1-000002B2  190'         **      200      1-000002C0  210'         **      220
  1-000002CE  230'         **      240      1-000002DC  250'     1-000002ED  260'         **      270      1-00000307  280'
      **      290      1-00000315  295'

FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name               Type  Name               Type  Name               Type  Name               Type  Name               Type  Name

   I*4  COMPRESS4           I*4  COMPRESSC           I*4  DECODE_ECC          I*4  LIB$EXTZV           I*4  LIB$LOCC                 LINCHK
        MA780_REGA               MOVC3                    OUTPUT
```

```
MS78

FUNC

  Ty

   I
```

```
0001
```

```
        Subroutine MS780C (lun,memory_registers)

c++
c       This routine displays the error log packet for the ms780c
c       memory controller.  The format of the packet is as follows.
c
c       +-----------------------------------+
c       |         memory register A         |
c       +-----------------------------------+
c       |         memory register B         |
c       +-----------------------------------+
c       |         memory register C         |
c       +-----------------------------------+
c--


        Implicit        none

        byte            lun

        integer*4       memory_registers
        integer*4       adapter_tr
        integer*4       buffer(3)
        integer*4       memory_register_a
        integer*4       memory_register_b
        integer*4       memory_register_c

        equivalence     (buffer(1),memory_register_a)
        equivalence     (buffer(2),memory_register_b)
        equivalence     (buffer(3),memory_register_c)

        character*27    memory_init_status(0:3)
        data            memory_init_status(0)   /'INITIALIZATION IN PROGRESS*'/
        data            memory_init_status(1)   /'MEMORY CONTAINS VALID DATA*'/
        data            memory_init_status(2)   /'INVALID STATE*'/
        data            memory_init_status(3)   /'INITIALIZATION COMPLETE*'/

        character*27    v2memory_registerb(14:14)
        data            v2memory_registerb(14)  /'START ADDRESS WRITE ENABLE*'/

        character*18    v1memory_registerc(28:30)
        data            v1memory_registerc(28)  /'ERROR LOG REQUEST*'/
        data            v1memory_registerc(29)  /'HIGH ERROR RATE*'/
        data            v1memory_registerc(30)  /'INHIBIT CRD TAG*'/

        integer*4       lib$extzv
        integer*4       I
        integer*4       decode_ecc
        integer*4       compress4
        integer*4       compressc
        integer*4       init_status
        integer*4       starting_address
        integer*4       error_syndrome
        integer*4       error_bit
        integer*4       error_bank_address
        integer*4       error_array
        logical*1       diagnostic_mode
        Logical*1       four_k
        Logical*1       sixteen_k


        call movc3 (%val(12),memory_registers,buffer)

        diagnostic_mode = .false.

        if (lib$extzv (8,2,memory_register_b) .ne. 0) diagnostic_mode = .true.

        if (.not. diagnostic_mode) then

        call ms780c_rega (lun,memory_register_a)
        else

        call linchk (lun,1)

        write(lun,5) memory_register_a
5       format(' ',t8,'CSRA',t24,z8.8)
        endif

        Four_k = .false.
        Sixteen_k = .false.

        If (LIB$EXTZV(3,2,memory_register_a) .EQ. 1) four_k = .true.
        If (LIB$EXTZV(3,2,memory_register_a) .EQ. 2) sixteen_k = .true.

        call linchk (lun,1)

        write(lun,10) 'CSRB',memory_register_b
10      format(' ',t8,a,t24,z8.8)

        if (diagnostic_mode) then

        call linchk (lun,1)

        write(lun,12) 'DIAGNOSTIC MODE'
12      format(' ',t40,a)
        endif

        if (.not. diagnostic_mode) then

        init_status = lib$extzv(12,2,memory_register_b)

        call linchk (lun,1)

        write(lun,30) memory_init_status(init_status)
30      format(' ',t40,a<compressc (memory_init_status(init_status))>)

        call output (lun,memory_register_b,v2memory_registerb,14,14,14,'0')

        starting_address = lib$extzv(15,13,memory_register_b)*64

        call linchk (lun,1)
        write(lun,35) starting_address
35      format(' ',t40,'MEMORY BASE ADDRESS = ',
        1 i<compress4 (starting_address)>,'.K')

        call linchk (lun,2)

        write(lun,40) (lib$extzv(i,2,memory_register_b),i = 28,30,2)
40      format(' ',t40,'FILE INPUT POINTER ',
        1 i<compress4 (lib$extzv(i,2,memory_register_b))>,'.',/,
        1 t40,'FILE OUTPUT POINTER ',
        1 i<compress4 (lib$extzv(i,2,memory_register_b))>,'.')
        endif

        call linchk (lun,1)

        write(lun,45) 'CSRC',memory_register_c
45      format(' ',t8,a,t24,z8.8)

        if (.not. diagnostic_mode) then

        if (lib$extzv(28,1,memory_register_c) .eq. 1) then

        error_syndrome = lib$extzv(0,8,memory_register_c)

        call linchk (lun,1)

        write(lun,15) error_syndrome
15      format(' ',t40,'ERROR SYNDROME = ',z2.2)

        error_bit = decode_ecc (error_syndrome,memory_register_a)

        call linchk (lun,1)

        if (error_bit .eq. -1) then

        write(lun,20) 'RDS ERROR'
20      format(' ',t40,a)
        else

        write(lun,25) 'CRD ERROR, CORRECTED BIT #',error_bit,'.'
25      format(' ',t40,a,i<compress4 (error_bit)>,a)
        endif

        If (four_k) then
        Error_bank_address = LIB$EXTZV(21,1,memory_register_c)

        Else if (sixteen_k) then
        Error_bank_address = LIB$EXTZV(23,1,memory_register_c)

        Endif

        call linchk (lun,1)

        write(lun,47) error_bank_address
47      format(' ',t40,'ARRAY BANK #',
        1 i<compress4 (error_bank_address)>,'. IN ERROR')
```

```
                                                      J 4
MS780C                                               16-Sep-1984 00:07:33    VAX-11 FORTRAN V3.4-56                Page 28
                                                      5-Sep-1984 14:01:18    DISK$VMSMASTER:[ERF.SRC]MEMORYS.FOR;1

                 error_array = lib$extzv(24,4,memory_register_c)

                 call linchk (lun,1)

                 write(lun,50) error_array
        50       format(' ',t40,'ARRAY #',i<compress4 (error_array)>,'. IN ERROR')
                 endif

                 call output (lun,memory_register_c,v1memory_registerc,28,28,30,'0')
                 endif

                 return
                 end


PROGRAM SECTIONS

    Name                                     Bytes   Attributes

  0 $CODE                                     1100   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                     411   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                     716   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

    Total Space Allocated                     2227

ENTRY POINTS

    Address   Type  Name

  0-00000000        MS780C

VARIABLES

    Address   Type  Name                                    Address   Type  Name

  2-000000CC  I*4   ADAPTER_TR                             2-000000C9  L*1   DIAGNOSTIC_MODE
  2-000000E8  I*4   ERROR_ARRAY                            2-000000E4  I*4   ERROR_BANK_ADDRESS
  2-000000E0  I*4   ERROR_BIT                              2-000000DC  I*4   ERROR_SYNDROME
  2-000000CA  L*1   FOUR_K                                 2-000000D0  I*4   I
  2-000000D4  I*4   INIT_STATUS                           AP-00000004@ L*1   LUN
 AP-00000008@ I*4   MEMORY_REGISTERS                       2-00000000  I*4   MEMORY_REGISTER_A
  2-00000004  I*4   MEMORY_REGISTER_B                      2-00000008  I*4   MEMORY_REGISTER_C
  2-000000CB  L*1   SIXTEEN_K                              2-000000D8  I*4   STARTING_ADDRESS

ARRAYS

    Address   Type  Name                                   Bytes  Dimensions

  2-00000000  I*4   BUFFER                                    12  (3)
  2-0000000C  CHAR  MEMORY_INIT_STATUS                       108  (0:3)
  2-00000093  CHAR  V1MEMORY_REGISTERC                        54  (28:30)
  2-00000078  CHAR  V2MEMORY_REGISTERB                        27  (14:14)
```

```
LABELS

    Address    Label         Address    Label         Address    Label         Address    Label         Address    Label         Address    Label

  1-00000079   5'          1-0000008A   10'         1-00000096   12'         1-00000123   15'         1-0000013F   20'         1-00000146   25'
  1-0000009D   30'         1-000000A9   35'         1-000000D1   40'         1-00000117   45'         1-00000154   47'         1-0000017A   50'


FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name                Type  Name                Type  Name                Type  Name                Type  Name                Type  Name

   I*4  COMPRESS4            I*4  COMPRESSC            I*4  DECODE_ECC           I*4  LIB$EXTZV                  LINCHK                     MOVC3
        MS780C_REGA               OUTPUT
```

```
0001
```

```
MS75

PROG

 0
 1
 2

ENTR

 0-

VARI

 2-
 2-
 2-
 2-
 2-
AP-
 2-
 2-

ARRA

 2-
 2-
 2-
 2-

LABE

 1-
 1-
 1-
```

```
        Subroutine MS780E (lun,memory_registers)

c++
c       This routine displays the error log packet for the ms780e
c       memory controller.  The format of the packet is as follows.
c
c       +-------------------------------------+
c       !          memory register A          !
c       +-------------------------------------+
c       !          memory register B          !
c       +-------------------------------------+
c       !          memory register C          !
c       +-------------------------------------+
c       !          memory register D          !
c       +-------------------------------------+
c--

        Implicit        none

        byte            lun

        integer*4       memory_registers
        integer*4       buffer(4)
        integer*4       memory_register_a
        integer*4       memory_register_b
        integer*4       memory_register('c'x:'d'x)

        equivalence     (buffer(1),memory_register_a)
        equivalence     (buffer(2),memory_register_b)
        equivalence     (buffer(3),memory_register)

        logical*1       diagnostic_mode

        integer*4       decode_ecc
        integer*4       compress4
        integer*4       compressc
        Integer*4       lib$extzv
        Integer*4       I

        character*27    memory_init_status(0:3)
        data            memory_init_status(0)   /'INITIALIZATION IN PROGRESS*'/
        data            memory_init_status(1)   /'MEMORY CONTAINS VALID DATA*'/
        data            memory_init_status(2)   /'INVALID STATE*'/
        data            memory_init_status(3)   /'INITIALIZATION COMPLETE*'/

        character*33    v1memory_registerb(7:7)
        data            v1memory_registerb(7)
        1 /'SBI INTERFACE WRITE PARITY ERROR*'/

        character*27    v2memory_registerb(14:14)
        data            v2memory_registerb(14)  /'START ADDRESS WRITE ENABLE*'/

        character*29    v1memory_register(7:7)
        data            v1memory_register(7)
        1 /'MICRO-SEQUENCER PARITY ERROR*'/

        character*18    v2memory_register(28:30)
```

MS75(

FUNC

Ty

I

```
                data            v2memory_register(28)   /'ERROR LOG REQUEST*'/
                data            v2memory_register(29)   /'HIGH ERROR RATE*'/
                data            v2memory_register(30)   /'INHIBIT CRD TAG*'/

                integer*4       init_status
                integer*4       starting_address
                integer*4       error_syndrome
                integer*4       error_bit
                integer*4       error_bank_address
                integer*4       error_array
                integer*4       adapter_tr


                call movc3 (%val(16),memory_registers,buffer)

                diagnostic_mode = .false.

                if (
                1 lib$extzv(7,3,memory_register_b) .ne. 0
                1 .or.
                1 lib$extzv(11,1,memory_register_b) .eq. 1
                1 .or.
                1 lib$extzv(31,1,memory_register('c'x)) .eq. 1
                1 .or.
                1 lib$extzv(31,1,memory_register('d'x)) .eq. 1
                1 ) then

                diagnostic_mode = .true.
                endif

                if (.not. diagnostic_mode) then

                call ms780e_rega (lun,memory_register_a)
                else

                call linchk (lun,1)

                write(lun,5) memory_register_a
         5      format(' ',t8,'CSRA',t24,z8.8)
                endif

                call linchk (lun,1)

                write(lun,10) 'CSRB',memory_register_b
        10      format(' ',t8,a,t24,z8.8)

                if (diagnostic_mode) then

                call linchk (lun,1)

                write(lun,12) 'DIAGNOSTIC MODE'
        12      format(' ',t40,a)
                endif

                if (.not. diagnostic_mode) then

                init_status = lib$extzv(12,2,memory_register_b)

        call linchk (lun,1)

        write(lun,15) memory_init_status(init_status)
15      format(' ',t40,a<compressc (memory_init_status(init_status))>)

        call output (lun,memory_register_b,v1memory_registerb,7,7,7,'0')

        call output (lun,memory_register_b,v2memory_registerb,14,14,14,'0')

        starting_address = lib$extzv(19,9,memory_register_b)

        call linchk (lun,1)

        write(lun,20) starting_address
20      format(' ',t40,'MEMORY BASE ADDRESS = ',
        1 i<compress4 (starting_address)>,'.M')
        endif

        do 55,i = 'c'x,'d'x

        call linchk (lun,1)

        if (i .eq. 'c'x) then

        write(lun,25) 'C',memory_register(i)
25      format(' ',t8,'CSR',a,t24,z8.8)

        else if (i .eq.'d'x) then

        write(lun,25) 'D',memory_register(i)
        endif

        if (.not. diagnostic_mode) then

        if (lib$extzv(28,1,memory_register(i)) .eq. 1) then

        error_syndrome = lib$extzv(0,7,memory_register(i))

        call linchk (lun,1)

        write(lun,30) error_syndrome
30      format(' ',t40,'ERROR SYNDROME = ',z2.2)

        error_bit = decode_ecc (error_syndrome,memory_register_a)

        call linchk (lun,1)

        if (error_bit .eq. -1) then

        write(lun,35) 'RDS ERROR'
35      format(' ',t40,a)
        else

        write(lun,40) 'CRD ERROR, CORRECTED BIT #',error_bit,'.'
40      format(' ',t40,a,i<compress4 (error_bit)>,a)
        endif
```

```

              call output (lun,memory_register(i),v1memory_register,7,7,7,'0')

              error_bank_address = lib$extzv(22,2,memory_register(i))

              call linchk (lun,1)

              write(lun,45) error_bank_address
       45     format(' ',t40,'ARRAY BANK #',
              1 i<compress4 (error_bank_address)>,'. IN ERROR')

              error_array = lib$extzv(24,3,memory_register(i))

              If (i .EQ. 'd'x) then
              Error_array = error_array + 8

              Endif

              call linchk (lun,1)

              write(lun,50) error_array
       50     format(' ',t40,'ARRAY #',i<compress4 (error_array)>,'. IN ERROR')
              endif

              call output (lun,memory_register(i),v2memory_register,28,28,30,'0')
              endif

       55     continue

              return
              end
```

```
                                                                D  5
MS780E                                                         16-Sep-1984 00:07:33    VAX-11 FORTRAN V3.4-56              Page  35
                                                                5-Sep-1984 14:01:18    DISK$VMSMASTER:[ERF.SRC]MEMORYS.FOR;1

PROGRAM SECTIONS

    Name                                    Bytes   Attributes

  0 $CODE                                    1098   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                    344   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                    888   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

    Total Space Allocated                    2330

ENTRY POINTS

    Address   Type  Name

  0-00000000        MS780E

VARIABLES

    Address   Type  Name                                          Address   Type  Name

  2-00000128  I*4   ADAPTER_TR                                  2-0000010B  L*1   DIAGNOSTIC_MODE
  2-00000124  I*4   ERROR_ARRAY                                 2-00000120  I*4   ERROR_BANK_ADDRESS
  2-0000011C  I*4   ERROR_BIT                                   2-00000118  I*4   ERROR_SYNDROME
  2-0000010C  I*4   I                                           2-00000110  I*4   INIT_STATUS
 AP-00000004@ L*1   LUN                                        AP-00000008@ I*4   MEMORY_REGISTERS
  2-00000000  I*4   MEMORY_REGISTER_A                           2-00000004  I*4   MEMORY_REGISTER_B
  2-00000114  I*4   STARTING_ADDRESS

ARRAYS

    Address   Type  Name                                        Bytes  Dimensions

  2-00000000  I*4   BUFFER                                         16  (4)
  2-00000010  CHAR  MEMORY_INIT_STATUS                            108  (0:3)
  2-00000008  I*4   MEMORY_REGISTER                                 8  (12:13)
  2-000000B8  CHAR  V1MEMORY_REGISTER                              29  (7:7)
  2-0000007C  CHAR  V1MEMORY_REGISTERB                             33  (7:7)
  2-000000D5  CHAR  V2MEMORY_REGISTER                              54  (28:30)
  2-0000009D  CHAR  V2MEMORY_REGISTERB                             27  (14:14)

LABELS

    Address   Label        Address   Label       Address   Label       Address   Label       Address   Label       Address   Label

  1-00000077  5'       1-00000088  10'      1-00000094  12'      1-0000009B  15'      1-000000A7  20'      1-000000CF  25'
  1-000000E0  30'      1-000000FC  35'      1-00000103  40'      1-00000111  45'      1-00000137  50'          **      55
```

```
MS780E                                            E  5
                                                16-Sep-1984 00:07:33    VAX-11 FORTRAN V3.4-56                     Page  36
                                                 5-Sep-1984 14:01:18    DISK$VMSMASTER:[ERF.SRC]MEMORYS.FOR;1

FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name             Type  Name              Type  Name              Type  Name              Type  Name              Type  Name

   I*4  COMPRESS4         I*4  COMPRESSC          I*4  DECODE_ECC         I*4  LIB$EXTZV               LINCHK                  MOVC3
        MS780E_REGA            OUTPUT
```

```
MS73

0173
0174
0175
0176
0177
0178
0179
0180
0181
0182
0183
0184
0185
0186
0187

PROG

  0
  1
  2

ENTR

 0-

VARI

  2-
  2-
  2-
 AP-
  2-
 AP-

ARRA

  2-
  2-
  2-
  2-
  2-
```

```
                                                                          F  5
                                                                        16-Sep-1984 00:07:33    VAX-11 FORTRAN V3.4-56               Page  37
                                                                         5-Sep-1984 14:01:18    DISK$VMSMASTER:[ERF.SRC]MEMORYS.FOR;1
0001
```

```
MS73
LABE
  1-
  1-
FUNC
  Ty
   I
```

```
        Subroutine MS750 (lun,memory_registers)

c++
c
c       Functional description:
c
c       This module displays ms750 memory error packets.  The format is as
c       follows.
c
c       +-------------------------------+
c       |       memory register 0       |
c       +-------------------------------+
c       |       memory register 1       |
c       +-------------------------------+
c       |       memory register 2       |
c       +-------------------------------+
c--

        Implicit        none

        byte            lun

        integer*4       memory_registers
        integer*4       slot_index
        integer*4       buffer(3)
        integer*4       memory_register_0
        integer*4       memory_register_1
        integer*4       memory_register_2

        equivalence     (buffer(1),memory_register_0)
        equivalence     (buffer(2),memory_register_1)
        equivalence     (buffer(3),memory_register_2)

        character*35    v1memory_register0(29:31)
        data            v1memory_register0(29)  /'CORRECTED ERROR FLAG*'/
        data            v1memory_register0(30)
        1 /'UNCORRECTED ERROR INFORMATION LOST*'/
        data            v1memory_register0(31)  /'UNCORRECTED ERROR FLAG*'/

        character*34    v1memory_register1(28:28)
        data            v1memory_register1(28)
        1 /'ENABLE REPORTING CORRECTED ERRORS*'/

        character*23    v1memory_register2(16:16)
        data            v1memory_register2(16)  /'BATTERY BACKUP FAILURE*'/

        integer*4       compress4
        integer*4       lib$extzv
        integer*4       I
        integer*4       decode_ecc
        integer*4       error_bit
        integer*4       error_array
        integer*4       error_bank
        integer*4       arrays_present
        integer*4       start_address

        equivalence     (error_bit,error_array,arrays_present,
             1 start_address)

             integer*4       array_code

             logical*1       diagnostic_mode
             logical*1       l0011
             logical*1       l0016


             call movc3 (%val(12),memory_registers,buffer)

             diagnostic_mode = .false.

             if (lib$extzv(25,3,memory_register_1) .ne. 0) diagnostic_mode = .true.

             call linchk (lun,2)

             write(lun,10) memory_register_0
     10      format(/' ',t8,'CSR0',t24,z8.8)

             if (.not. diagnostic_mode) then

             call linchk (lun,1)

             write(lun,15) lib$extzv(0,7,memory_register_0)
     15      format(' ',t40,'ERROR SYNDROME = ',z2.2)

             if (lib$extzv(29,1,memory_register_0) .eq. 1) then

             error_bit = decode_ecc (lib$extzv(0,7,memory_register_0))

             call linchk (lun,1)

             if (error_bit .eq. -1) then

             write(lun,20) '"ECC" CODE, UNCORRECTED ERROR'
             else

             write(lun,20) 'CORRECTED ERROR, BIT #',error_bit,'.'
     20      format(' ',t40,a,:i<compress4 (error_bit)>,:a)
             endif
             endif

             if (lib$extzv(30,1,memory_register_0) .eq. 0) then

             error_array = lib$extzv(9,15,memory_register_0)

             l0011 = .false.

             l0016 = .false.

             if (lib$extzv(24,1,memory_register_2) .eq. 1) then

             l0016 = .true.
             else

             l0011 = .true.
```

```
                endif

                do 25,i = 0,15,2

                array_code = lib$extzv(i,2,memory_register_2)

                if (l0016) then

                if (array_code .eq. 3) then

                error_array = error_array - 512

                If (LIB$EXTZV(17,1,memory_register_0) .EQ. 0) then
                Error_bank = 0

                Else
                Error_bank = 1

                Endif

                else if (array_code .eq. 2) then

                error_array = error_array - 2048
                Error_bank = LIB$EXTZV(19,2,memory_register_0)

                endif

                else if (l0011) then

                if (array_code .eq. 1) then

                error_array = error_array - 256
                Error_bank = 0

                else if (array_code .eq. 3) then

                error_array = error_array - 512

                If (LIB$EXTZV(17,1,memory_register_0) .EQ. 0) then
                Error_bank = 0

                Else
                Error_bank = 1

                Endif
                endif
                endif

                if (error_array .le. 0) then

                error_array = i/2

                goto 26
                endif

        25      continue

```

```
MEMO

PROG

  0
  1
  2

ENTR

  0-

VARI

  2-
  2-

ARRA

 AP-
  2-
  2-
  2-
  2-
```

```
26      call linchk (lun,1)

        Write (lun,28) error_bank
28      Format(' ',T40,'ARRAY BANK #',
        1 I<COMPRESS4 (error_bank)>,'. IN ERROR')

        Call LINCHK (lun,1)

        write(lun,30) error_array
30      format(' ',t40,'ARRAY #',i<compress4 (error_array)>,'. IN ERROR')
        endif

        call output (lun,memory_register_0,v1memory_register0,29,29,31,'0')
        endif

        call linchk (lun,1)

        write(lun,35) memory_register_1
35      format(' ',t8,'CSR1',t24,z8.8)

        if (lib$extzv(25,3,memory_register_1) .eq. 0) then

        call output (lun,memory_register_1,v1memory_register1,28,28,28,'0')
        else

        call linchk (lun,1)

        write(lun,40) 'DIAGNOSTIC MODE'
40      format(' ',t40,a)
        endif

        call linchk (lun,1)

        write(lun,45) memory_register_2
45      format(' ',t8,'CSR2',t24,z8.8)

        if (.not. diagnostic_mode) then

        arrays_present = 0

        do 47,i = 0,15,2

        array_code = lib$extzv(i,2,memory_register_2)

        if (L0016) then

        if (array_code .eq. 3) then

        arrays_present = arrays_present + 2

        else if (array_code .eq. 2) then

        arrays_present = arrays_present + 8
        endif

        else if (L0011) then
```

```
          if (array_code .eq. 1) then

          arrays_present = arrays_present + 1

          else if (array_code .eq. 3) then

          arrays_present = arrays_present + 2
          endif
          endif

  47      continue

          call linchk (lun,1)

          write(lun,50) arrays_present*128
  50      format(' ',t40,'MEMORY SIZE = ',
          1 i<compress4 (arrays_present*128)>,'.K')

          call output (lun,memory_register_2,v1memory_register2,16,16,16,'0')

          start_address = lib$extzv (17,7,memory_register_2)

          call linchk (lun,1)

          write(lun,55) start_address*128
  55      format(' ',t40,'MEMORY BASE ADDRESS = ',
          1 i<compress4 (start_address*128)>,'.K')
          endif

          call linchk (lun,1)

          if (L0016) then

          write(lun,60) 'L0016'
  60      format(' ',t40,'CONTROLLER IS ',a)
          else

          write(lun,60) 'L0011'
          endif

          return
          end
```

0001

```
                                                                    L  5
MS750                                                        16-Sep-1984 00:07:33    VAX-11 FORTRAN V3.4-56                  Page  43
                                                              5-Sep-1984 14:01:18    DISK$VMSMASTER:[ERF.SRC]MEMORYS.FOR;1

PROGRAM SECTIONS

    Name                                         Bytes   Attributes

  0 $CODE                                         1271   PIC CON REL LCL   SHR   EXE    RD NOWRT LONG
  1 $PDATA                                         412   PIC CON REL LCL   SHR NOEXE    RD NOWRT LONG
  2 $LOCAL                                         680   PIC CON REL LCL NOSHR NOEXE    RD   WRT LONG

    Total Space Allocated                         2363

ENTRY POINTS

    Address   Type  Name

  0-00000000        MS750

VARIABLES

    Address   Type  Name                                       Address   Type  Name

  2-00000000  I*4   ARRAYS_PRESENT                           2-000000C4  I*4   ARRAY_CODE
  2-000000B2  L*1   DIAGNOSTIC_MODE                          2-00000000  I*4   ERROR_ARRAY
  2-000000C0  I*4   ERROR_BANK                               2-00000000  I*4   ERROR_BIT
  2-000000BC  I*4   I                                        2-000000B3  L*1   L0011
  2-000000B4  L*1   L0016                                   AP-00000004@ L*1   LUN
 AP-00000008@ I*4   MEMORY_REGISTERS                         2-00000004  I*4   MEMORY_REGISTER_0
  2-00000008  I*4   MEMORY_REGISTER_1                        2-0000000C  I*4   MEMORY_REGISTER_2
  2-000000B8  I*4   SLOT_INDEX                               2-00000000  I*4   START_ADDRESS

ARRAYS

    Address   Type  Name                                          Bytes  Dimensions

  2-00000004  I*4   BUFFER                                           12  (3)
  2-00000010  CHAR  V1MEMORY_REGISTER0                              105  (29:31)
  2-00000079  CHAR  V1MEMORY_REGISTER1                               34  (28:28)
  2-0000009B  CHAR  V1MEMORY_REGISTER2                               23  (16:16)

LABELS

    Address   Label       Address   Label       Address   Label       Address   Label       Address   Label       Address   Label

  1-0000008F  10'       1-000000A1  15'       1-000000BD  20'             **    25        0-0000022E  26        1-000000CD  28'
  1-000000F3  30'       1-00000114  35'       1-00000125  40'       1-0000012C  45'             **    47        1-0000013D  50'
  1-0000015D  55'       1-00000185  60'
```

```
                                                                    M  5
MS750                                                              16-Sep-1984 00:07:33     VAX-11 FORTRAN V3.4-56                  Page  44
                                                                    5-Sep-1984 14:01:18     DISK$VMSMASTER:[ERF.SRC]MEMORYS.FOR;1

FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name                 Type  Name                 Type  Name                 Type  Name                 Type  Name                 Type  Name

   I*4  COMPRESS4             I*4  DECODE_ECC            I*4  LIB$EXTZV                  LINCHK                     MOVC3                      OUTPUT
```

MEMO

```
        Subroutine MS730 (lun,memory_register_0)

c**
c       Functional description:
c
c       This routine displays the memory registers for the 11/730.  The
c       format of the buffer is as follows.
c
c       +---------------------------------+
c       !           memory csr0           !
c       +---------------------------------+
c       !           memory csr1           !
c       +---------------------------------+
c       !           memory csr2           !
c       +---------------------------------+
c--

        Implicit        none

        byte            lun

        integer*4       memory_register_0
        integer*4       buffer(3)
        integer*4       memory_csr0
        integer*4       memory_csr1
        integer*4       memory_csr2

        equivalence     (buffer(1),memory_csr0)
        equivalence     (buffer(2),memory_csr1)
        equivalence     (buffer(3),memory_csr2)

        logical*1       diagnostic_mode

        integer*4       compress4
        integer*4       decode_ecc
        integer*4       error_bit
        integer*4       error_array
        integer*4       error_bank
        integer*4       kilo_bytes_present
        integer*4       lib$extzv
        integer*4       I

        character*23    v1memory_register_1(27:28)
        data            v1memory_register_1(27) /'MEMORY MAPPING ENABLE*'/
        data            v1memory_register_1(28) /'ENABLE "CRD" REPORTING*'/

        character*12    v2memory_register_1(30:30)
        data            v2memory_register_1(30) /'"CRD" ERROR*'/

        character*17    v1memory_register_2(24:24,0:1)
        data            v1memory_register_2(24,0) /'16K RAMS PRESENT*'/
        data            v1memory_register_2(24,1) /'64K RAMS PRESENT*'/


        call movc3 (%val(12),memory_register_0,buffer)
```

```
                diagnostic_mode = .false.

                if (iand(memory_csr1,'26000000'x) .ne. 0) diagnostic_mode = .true.

                call linchk (lun,2)

                write(lun,10) memory_csr0
10              format(/' ',t8,'CSR0',t24,z8.8)

                if (.not. diagnostic_mode) then

c
c               11/730 syndrome bits are inverted so...
c

                call linchk (lun,1)

                write(lun,15) lib$extzv(0,7,not(lib$extzv(0,7,memory_csr0)))
15              format(' ',t40,'ERROR SYNDROME = ',z2.2)

                error_bit = decode_ecc (lib$extzv(0,7,not(lib$extzv(0,7,memory_csr0))))

                call linchk (lun,1)

                if (error_bit .eq. -1) then

                write(lun,20) '''ECC'' CODE, UNCORRECTED ERROR'
                else

                write(lun,20) 'CORRECTED ERROR, BIT #',error_bit,'.'
20              format(' ',t40,a,:i<compress4 (error_bit)>,:a)
                endif

                error_array = lib$extzv(9,15,memory_csr0)

                If (LIB$EXTZV(24,1,memory_csr2) .EQ. 1) then
                Error_bank = LIB$EXTZV(19,2,memory_csr0)

                Else
                Error_bank = LIB$EXTZV(17,1,memory_csr0)

                Endif

                Call LINCHK (lun,1)
                Write (lun,22) error_bank
22              Format(' ',T40,'ARRAY BANK #',
                1 I<COMPRESS4 (error_bank)>,'. IN ERROR')

                do 25,i = 0,15

                if (lib$extzv(i,1,memory_csr2) .eq. 1) then

                if (lib$extzv(24,1,memory_csr2) .eq. 1) then

                error_array = error_array - 1024
                else

```

```
            error_array = error_array - 256
            endif
            endif

            if (error_array .le. 0) then

            error_array = i/2

            goto 27
            endif

 25         continue

 27         call linchk (lun,1)

            write(lun,30) error_array
 30         format(' ',t40,'ARRAY #',i<compress4 (error_array)>,'. IN ERROR')
            endif

            call linchk (lun,1)

            write(lun,35) memory_csr1
 35         format(' ',t8,'CSR1',t24,z8.8)

            if (.not. diagnostic_mode) then

            call output (lun,memory_csr1,v1memory_register_1,27,27,28,'0')

            call output (lun,memory_csr1,v2memory_register_1,30,30,30,'0')
            else

            call linchk (lun,1)

            write(lun,40) 'DIAGNOSTIC MODE'
 40         format(' ',t40,a)
            endif

            call linchk (lun,1)

            write(lun,45) memory_csr2
 45         format(' ',t8,'CSR2',t24,z8.8)

            if (.not. diagnostic_mode) then

            kilo_bytes_present = 0

            do 50,i = 0,15

            if (lib$extzv(i,1,memory_csr2) .eq. 1) then

            if (lib$extzv(24,1,memory_csr2) .eq. 1) then

            kilo_bytes_present = kilo_bytes_present + 512
            else

            kilo_bytes_present = kilo_bytes_present + 128
            endif
```

```
                endif

        50      continue

                call linchk (lun,1)

                write(lun,55) kilo_bytes_present
        55      format(' ',t40,'MEMORY SIZE = ',i<compress4 (kilo_bytes_present)>,
                1 ' K')

                call output (lun,memory_csr2,v1memory_register_2,24,24,24,'2')
                endif

                return
                end


PROGRAM SECTIONS

    Name                                            Bytes   Attributes

  0 $CODE                                             925   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                            325   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                            572   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

    Total Space Allocated                            1822

ENTRY POINTS

    Address   Type  Name

  0-00000000        MS730

VARIABLES

    Address   Type  Name                                      Address   Type  Name

  2-00000068  L*1   DIAGNOSTIC_MODE                         2-00000070  I*4   ERROR_ARRAY
  2-00000074  I*4   ERROR_BANK                              2-0000006C  I*4   ERROR_BIT
  2-0000007C  I*4   I                                       2-00000078  I*4   KILO_BYTES_PRESENT
 AP-00000004@ L*1   LUN                                     2-00000000  I*4   MEMORY_CSR0
  2-00000004  I*4   MEMORY_CSR1                             2-00000008  I*4   MEMORY_CSR2
 AP-00000008@ I*4   MEMORY_REGISTER_0

ARRAYS

    Address   Type  Name                                       Bytes  Dimensions

  2-00000000  I*4   BUFFER                                        12  (3)
  2-0000000C  CHAR  V1MEMORY_REGISTER_1                           46  (27:28)
  2-00000046  CHAR  V1MEMORY_REGISTER_2                           34  (24:24, 0:1)
  2-0000003A  CHAR  V2MEMORY_REGISTER_1                           12  (30:30)
```

```
MS730                                            F  6
                                                16-Sep-1984 00:07:33    VAX-11 FORTRAN V3.4-56                     Page  50
                                                 5-Sep-1984 14:01:18    DISK$VMSMASTER:[ERF.SRC]MEMORYS.FOR;1

LABELS

   Address    Label       Address    Label       Address    Label       Address    Label       Address    Label       Address    Label

 1-00000077  10'       1-00000089  15'       1-000000A5  20'       1-000000B5  22'            **       25          0-000001FE  27
 1-000000DB  30'       1-000000FC  35'       1-0000010D  40'       1-00000114  45'            **       50          1-00000125  55'

FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name                Type  Name                Type  Name                Type  Name                Type  Name                Type  Name

   I*4  COMPRESS4            I*4  DECODE_ECC           I*4  LIB$EXTZV                 LINCHK                    MOVC3                     OUTPUT
```

```
0001
0002
```

```
              Subroutine MEMORY_REGISTER_UV1 (lun,memory_registers)

        C++
        C     This routine displays the UVAX1 memory registers.
        C     The format of the memory sub packet is as follows:
        C
        C     +-----------------------------------+
        C     |             CSR count             |
        C     +-----------------------------------+
        C     |            CSR or zero            |
        C     +-----------------------------------+
        C     .                                   .
        C     .                                   .
        C     .                                   .
        C     +-----------------------------------+
        C     |            CSR or zero            |
        C     +-----------------------------------+
        C--

              Implicit        none

              byte            lun

              integer*4       memory_registers(0:16)
              integer*4       error_address1
              integer*4       error_address2
              integer*4       loop
              integer*4       compress4
              integer*4       lib$extzv
              integer*4       lib$insv

              character*20    v1csr(0:0)
              data            v1csr(0)        /'PARITY ERROR ENABLE*'/

              character*16    v2csr(2:2)
              data            v2csr(2)        /'DIAGNOSTIC MODE*'/

              character*25    v3csr(14:15)
              data            v3csr(14)       /'EXTENDED CSR READ ENABLE*'/
              data            v3csr(15)       /'PARITY ERROR*'/

              call linchk (lun,1)
              Write (lun,1)
        1     Format (' ')

              do 15,loop = 1,16

              if (Lib$extzv(15,1,memory_registers(loop)) .eq. 1) then

              call linchk (lun,1)

              write(lun,5) loop,memory_registers(loop)
        5     format(' ',T8,'CSR #',i<compress4 (loop)>,t24,z8.8)

              call output (lun,memory_registers(loop),v1csr,0,0,0,'0')
              call output (lun,memory_registers(loop),v2csr,2,2,2,'0')
```

```
                error_address1 = lib$extzv (5,7,memory_registers(loop))
                error_address1 = lib$insv (error_address1,11,7,error_address1)

                error_address2 = lib$extzv (5,4,memory_registers(loop))
                error_address1 = lib$insv (error_address2,18,4,error_address1)

                call linchk (lun,1)

                write(lun,10) error_address1
        10      format(' ',t40,'PARITY ERROR ADDRESS, ',
                1 i<compress4 (error_address1)>,'.K')

                call output (lun,memory_registers(loop),v3csr,14,14,15,'0')
                endif

        15      continue

                Return
                End


PROGRAM SECTIONS

    Name                                     Bytes   Attributes

  0 $CODE                                      383   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                     110   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                     436   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

    Total Space Allocated                      929


ENTRY POINTS

    Address   Type  Name

  0-00000000        MEMORY_REGISTER_UV1


VARIABLES

    Address   Type  Name                                   Address   Type  Name

  2-00000058  I*4   ERROR_ADDRESS1                       2-0000005C  I*4   ERROR_ADDRESS2
  2-00000060  I*4   LOOP                                AP-00000004@ L*1   LUN


ARRAYS

    Address   Type  Name                                      Bytes  Dimensions

 AP-00000008@ I*4   MEMORY_REGISTERS                             68  (0:16)
  2-00000000  CHAR  V1CSR                                        20  (0:0)
  2-00000014  CHAR  V2CSR                                        16  (2:2)
  2-00000024  CHAR  V3CSR                                        50  (14:15)
```

```
LABELS

   Address    Label       Address    Label       Address    Label       Address    Label

  1-0000002A  1'        1-0000002E  5'         1-00000046  10'           **       15

FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name            Type  Name            Type  Name            Type  Name            Type  Name

   I*4  COMPRESS4        I*4  LIB$EXTZV        I*4  LIB$INSV              LINCHK                OUTPUT
```

```
0001
```

```
C++
C       Functional description:
C
C       This module maintains a list which is used to produce a display
C       that shows where and how many memory errors have occured.  The format
C       of the list is shown below.
C
C       +-------------------------------+
C       |            flink1             |
C       +-------------------------------+
C       |            blink1             |
C       +-------------------------------+
C       |          logging sid          |
C       +-------------------------------+
C       |    root node memory flink     |
C       +-------------------------------+
C       |    root memory node blink     |
C       +-------------------------------+
C       |    memory node entry count    |
C       +-------------------------------+
C
C
C       +-------------------------------+
C       |            flink2             |
C       +-------------------------------+
C       |            blink2             |
C       +-------------------------------+
C       |         memory node           |
C       +-------------------------------+
C       |       rout array flink        |
C       +-------------------------------+
C       |       root array blink        |
C       +-------------------------------+
C       |       array entry count       |
C       +-------------------------------+
C
C
C       +-------------------------------+
C       |            flink3             |
C       +-------------------------------+
C       |            blink3             |
C       +-------------------------------+
C       |            array              |
C       +-------------------------------+
C       |    root array bank flink      |
C       +-------------------------------+
C       |    root array bank blink      |
C       +-------------------------------+
C       |    array bank entry count     |
C       +-------------------------------+
C
C
C       +-------------------------------+
C       |            flink4             |
C       +-------------------------------+
C       |            blink4             |
C       +-------------------------------+
C               !          array bank          !
C               +------------------------------+
C               !     root array bit flink     !
C               +------------------------------+
C               !     root array bit blink     !
C               +------------------------------+
C               !   array page entry count     !
C               +------------------------------+
C
C
C               +------------------------------+
C               !            flink5            !
C               +------------------------------+
C               !            blink5            !
C               +------------------------------+
C               !          array bit           !
C               +------------------------------+
C               !         error count          !
C               +------------------------------+
C--

        Subroutine MEMORY_0 (search_sid,search_memory_node,
        1 search_array,search_array_bank,search_array_bit)

        Implicit        none

        byte            lun

        integer*4       buffer0(2)
        integer*4       buffer1(6)
        integer*4       buffer2(6)
        integer*4       buffer3(6)
        integer*4       buffer4(6)
        integer*4       buffer5(4)
        integer*4       root_logging_sid_flink
        integer*4       root_logging_sid_blink

        equivalence     (buffer0(1),root_logging_sid_flink)
        equivalence     (buffer0(2),root_logging_sid_blink)

        integer*4       flink1
        integer*4       blink1
        integer*4       logging_sid
        integer*4       root_memory_node_flink
        integer*4       root_memory_node_blink
        integer*4       memory_node_entry_count

        equivalence     (buffer1(1),flink1)
        equivalence     (buffer1(2),blink1)
        equivalence     (buffer1(3),logging_sid)
        equivalence     (buffer1(4),root_memory_node_flink)
        equivalence     (buffer1(5),root_memory_node_blink)
        equivalence     (buffer1(6),memory_node_entry_count)

        integer*4       flink2
        integer*4       blink2
        integer*4       memory_node
```

```
          integer*4       root_array_flink
          integer*4       root_array_blink
          integer*4       array_entry_count

          equivalence     (buffer2(1),flink2)
          equivalence     (buffer2(2),blink2)
          equivalence     (buffer2(3),memory_node)
          equivalence     (buffer2(4),root_array_flink)
          equivalence     (buffer2(5),root_array_blink)
          equivalence     (buffer2(6),array_entry_count)

          integer*4       flink3
          integer*4       blink3
          integer*4       array
          integer*4       root_array_bank_flink
          integer*4       root_array_bank_blink
          integer*4       array_bank_entry_count

          equivalence     (buffer3(1),flink3)
          equivalence     (buffer3(2),blink3)
          equivalence     (buffer3(3),array)
          equivalence     (buffer3(4),root_array_bank_flink)
          equivalence     (buffer3(5),root_array_bank_blink)
          equivalence     (buffer3(6),array_bank_entry_count)

          integer*4       flink4
          integer*4       blink4
          integer*4       array_bank
          integer*4       root_array_bit_flink
          integer*4       root_array_bit_blink
          integer*4       array_bit_entry_count

          equivalence     (buffer4(1),flink4)
          equivalence     (buffer4(2),blink4)
          equivalence     (buffer4(3),array_bank)
          equivalence     (buffer4(4),root_array_bit_flink)
          equivalence     (buffer4(5),root_array_bit_blink)
          equivalence     (buffer4(6),array_bit_entry_count)

          integer*4       flink5
          integer*4       blink5
          integer*4       array_bit
          integer*4       error_count

          equivalence     (buffer5(1),flink5)
          equivalence     (buffer5(2),blink5)
          equivalence     (buffer5(3),array_bit)
          equivalence     (buffer5(4),error_count)

          integer*4       insert_blink
          integer*4       logging_sid_entry_count
          integer*4       logging_sid_entry_address
          integer*4       memory_node_entry_address
          integer*4       array_entry_address
          integer*4       array_bank_entry_address
          integer*4       array_bit_entry_address
          integer*4       search_sid
```

```
        integer*4       search_memory_node
        integer*4       search_array
        integer*4       search_array_bank
        integer*4       search_array_bit
        integer*4       compress4
        Integer*4       I
        Integer*4       J
        Integer*4       K
        Integer*4       L
        Integer*4       M
        Integer*4       lib$extzv
        Integer*4       Root_flink
        Integer*4       Sid_count
        Integer*4       Node_count
        Integer*4       Array_count
        Integer*4       Bank_count
        Integer*4       Bit_count

        logical*1       lib$get_vm


        logging_sid_entry_address = root_logging_sid_flink

        do 100,i = 1,logging_sid_entry_count

        call movc3 (%val(24),%val(logging_sid_entry_address),buffer1)

        if (search_sid .eq. logging_sid) then

5       memory_node_entry_address = root_memory_node_flink

        do 90,j = 1,memory_node_entry_count

        call movc3 (%val(24),%val(memory_node_entry_address),buffer2)

        if (search_memory_node .eq. memory_node) then

10      array_entry_address = root_array_flink

        do 80,k = 1,array_entry_count

        call movc3 (%val(24),%val(array_entry_address),buffer3)

        if (search_array .eq. array) then

15      array_bank_entry_address = root_array_bank_flink

        do 70,l = 1,array_bank_entry_count

        call movc3 (%val(24),%val(array_bank_entry_address),buffer4)

        if (search_array_bank .eq. array_bank) then

20      array_bit_entry_address = root_array_bit_flink

        do 60,m = 1,array_bit_entry_count

```

```
                call movc3 (%val(16),%val(array_bit_entry_address),buffer5)

                if (search_array_bit .eq. array_bit) then

        25      error_count = error_count + 1

                call movl (error_count,%val(array_bit_entry_address + 12))

                return
                endif

                array_bit_entry_address = flink5

        60      continue

                call movc5 (%val(0),,%val(0),%val(16),buffer5)

                if (lib$get_vm(((16+7)/8)*8,array_bit_entry_address)) then

                call insque (%val(array_bit_entry_address),
                1 %val(root_array_bit_blink))

                array_bit = search_array_bit

                call movq (array_bit,%val(array_bit_entry_address + 8))

                array_bit_entry_count = array_bit_entry_count + 1

                call movl (array_bit_entry_count,%val(array_bank_entry_address + 20))

                goto 25
                endif

                return
                endif

                insert_blink = blink4

                if (array_bank .gt. search_array_bank) goto 75

                array_bank_entry_address = flink4

        70      continue

                insert_blink = root_array_bank_blink

        75      call movc5 (%val(0),,%val(0),%val(24),buffer4)

                if (lib$get_vm(((24+7)/8)*8,array_bank_entry_address)) then

                call insque (%val(array_bank_entry_address),%val(insert_blink))

                array_bank = search_array_bank

                root_array_bit_flink = array_bank_entry_address + 12

                root_array_bit_blink = root_array_bit_flink
                call movc3 (%val(16),array_bank,%val(array_bank_entry_address + 8))

                array_bank_entry_count = array_bank_entry_count + 1

                call movl (array_bank_entry_count,%val(array_entry_address + 20))

                goto 20
                endif

                return
                endif

                insert_blink = blink3

                if (array .gt. search_array) goto 85

                array_entry_address = flink3

        80      continue

                insert_blink = root_array_blink

        85      call movc5 (%val(0),,%val(0),%val(24),buffer3)

                if (lib$get_vm(((24+7)/8)*8,array_entry_address)) then

                call insque (%val(array_entry_address),%val(insert_blink))

                array = search_array

                root_array_bank_flink = array_entry_address + 12

                root_array_bank_blink = root_array_bank_flink

                call movc3 (%val(16),array,%val(array_entry_address + 8))

                array_entry_count = array_entry_count + 1

                call movl (array_entry_count,%val(memory_node_entry_address + 20))

                goto 15
                endif

                return
                endif

                insert_blink = blink2

                if (memory_node .gt. search_memory_node) goto 95

                memory_node_entry_address = flink2

        90      continue

                insert_blink = root_memory_node_blink
        95      call movc5 (%val(0),,%val(0),%val(24),buffer2)

                if (lib$get_vm(((24+7)/8)*8,memory_node_entry_address)) then

                call insque (%val(memory_node_entry_address),%val(insert_blink))

                memory_node = search_memory_node

                root_array_flink = memory_node_entry_address + 12

                root_array_blink = root_array_flink

                call movc3 (%val(16),memory_node,%val(memory_node_entry_address + 8))

                memory_node_entry_count = memory_node_entry_count + 1

                call movl (memory_node_entry_count,
                1 %val(logging_sid_entry_address + 20))

                goto 10
                endif

                return
                endif

                logging_sid_entry_address = flink1

        100     continue

                call movc5 (%val(0),,%val(0),%val(24),buffer1)

                if (logging_sid_entry_count .eq. 0) then

                root_logging_sid_flink = %loc(root_logging_sid_flink)

                root_logging_sid_blink = root_logging_sid_flink
                endif

                if (lib$get_vm(((24+7)/8)*8,logging_sid_entry_address)) then

                call insque (%val(logging_sid_entry_address),
                1 %val(root_logging_sid_blink))

                logging_sid = search_sid

                root_memory_node_flink = logging_sid_entry_address + 12

                root_memory_node_blink = root_memory_node_flink

                call movc3 (%val(16),logging_sid,%val(logging_sid_entry_address + 8))

                logging_sid_entry_count = logging_sid_entry_count + 1

                goto 5
                endif

                return
```

```
C
C This routine returns the root flink and the number of entries for the
C memory information queue.
C
        Entry GET_MEMORY_Q_INFO (root_flink,sid_count,node_count,array_count,
        1 bank_count,bit_count)

        Root_flink = root_logging_sid_flink
        Sid_count = logging_sid_entry_count
        Node_count = memory_node_entry_count
        Array_count = array_entry_count
        Bank_count = array_bank_entry_count
        Bit_count = array_bit_entry_count

        Return

        End


PROGRAM SECTIONS

    Name                                   Bytes   Attributes

  0 $CODE                                    925   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                     8   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                   608   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

    Total Space Allocated                   1541

ENTRY POINTS

    Address   Type  Name                              Address   Type  Name

  0-00000375        GET_MEMORY_Q_INFO               0-00000000        MEMORY_Q

VARIABLES

    Address   Type  Name                              Address   Type  Name

  2-00000030  I*4   ARRAY                           2-00000018  I*4   ARRAY_BANK
  2-00000090  I*4   ARRAY_BANK_ENTRY_ADDRESS        2-0000003C  I*4   ARRAY_BANK_ENTRY_COUNT
  2-00000008  I*4   ARRAY_BIT                       2-00000094  I*4   ARRAY_BIT_ENTRY_ADDRESS
  2-00000024  I*4   ARRAY_BIT_ENTRY_COUNT          AP-00000010@ I*4   ARRAY_COUNT
  2-0000008C  I*4   ARRAY_ENTRY_ADDRESS             2-00000054  I*4   ARRAY_ENTRY_COUNT
 AP-00000014@ I*4   BANK_COUNT                     AP-00000018@ I*4   BIT_COUNT
  2-0000005C  I*4   BLINK1                          2-00000044  I*4   BLINK2
  2-0000002C  I*4   BLINK3                          2-00000014  I*4   BLINK4
  2-00000004  I*4   BLINK5                          2-00000098  I*4   COMPRESS4
  2-0000000C  I*4   ERROR_COUNT                     2-00000058  I*4   FLINK1
  2-00000040  I*4   FLINK2                          2-00000028  I*4   FLINK3
  2-00000010  I*4   FLINK4                          2-00000000  I*4   FLINK5
  2-0000009C  I*4   I                               2-0000007C  I*4   INSERT_BLINK
```

```
MEMORY_Q                                               6 7
                                                      16-Sep-1984 00:07:33    VAX-11 FORTRAN V3.4-56                  Page  64
                                                       5-Sep-1984 14:01:18    DISK$VMSMASTER:[ERF.SRC]MEMORYS.FOR;1

  2-000000A0   I*4   J                                   2-000000A4   I*4   K
  2-000000A8   I*4   L                                   2-000000B0   I*4   LIB$EXTZV
  2-00000060   I*4   LOGGING_SID                         2-00000084   I*4   LOGGING_SID_ENTRY_ADDRESS
  2-00000080   I*4   LOGGING_SID_ENTRY_COUNT             2-00000078   L*1   LUN
  2-000000AC   I*4   M                                   2-00000048   I*4   MEMORY_NODE
  2-00000088   I*4   MEMORY_NODE_ENTRY_ADDRESS           2-0000006C   I*4   MEMORY_NODE_ENTRY_COUNT
 AP-0000000C@  I*4   NODE_COUNT                          2-00000038   I*4   ROOT_ARRAY_BANK_BLINK
  2-00000034   I*4   ROOT_ARRAY_BANK_FLINK               2-00000020   I*4   ROOT_ARRAY_BIT_BLINK
  2-0000001C   I*4   ROOT_ARRAY_BIT_FLINK                2-00000050   I*4   ROOT_ARRAY_BLINK
  2-0000004C   I*4   ROOT_ARRAY_FLINK                   AP-00000004@  I*4   ROOT_FLINK
  2-00000074   I*4   ROOT_LOGGING_SID_BLINK              2-00000070   I*4   ROOT_LOGGING_SID_FLINK
  2-00000068   I*4   ROOT_MEMORY_NODE_BLINK              2-00000064   I*4   ROOT_MEMORY_NODE_FLINK
 AP-0000000C@  I*4   SEARCH_ARRAY                       AP-00000010@  I*4   SEARCH_ARRAY_BANK
 AP-00000014@  I*4   SEARCH_ARRAY_BIT                   AP-00000008@  I*4   SEARCH_MEMORY_NODE
 AP-00000004@  I*4   SEARCH_SID                         AP-00000008@  I*4   SID_COUNT

ARRAYS

    Address   Type  Name                   Bytes  Dimensions

  2-00000070  I*4   BUFFER0                    8  (2)
  2-00000058  I*4   BUFFER1                   24  (6)
  2-00000040  I*4   BUFFER2                   24  (6)
  2-00000028  I*4   BUFFER3                   24  (6)
  2-00000010  I*4   BUFFER4                   24  (6)
  2-00000000  I*4   BUFFER5                   16  (4)

LABELS

    Address   Label       Address   Label       Address   Label       Address   Label       Address   Label       Address   Label

  0-0000003A  5       0-0000006B  10      0-0000009C  15      0-000000CD  20      0-000000F9  25          **      60
      **      70      0-00000194  75          **      80      0-00000218  85          **      90      0-0000029C  95
      **      100

FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name           Type  Name           Type  Name           Type  Name           Type  Name           Type  Name

        INSQUE         L*1   LIB$GET_VM           MOVC3                MOVC5                MOVL                 MOVQ
```

```
                                        H  7
                                        16-Sep-1984 00:07:33    VAX-11 FORTRAN V3.4-56                  Page  65
                                         5-Sep-1984 14:01:18    DISK$VMSMASTER:[ERF.SRC]MEMORYS.FOR;1
```

```
C++
C       Functional description:
C
C       This entry point is used to display the memory error occurrance list
C       built by memory_q.
C--

        Subroutine MEMORY_DISPLAY (lun)

        Implicit        none

        integer*4       buffer0(2)
        integer*4       buffer1(6)
        integer*4       buffer2(6)
        integer*4       buffer3(6)
        integer*4       buffer4(6)
        integer*4       buffer5(4)
        integer*4       root_logging_sid_flink
        integer*4       root_logging_sid_blink

        equivalence     (buffer0(1),root_logging_sid_flink)
        equivalence     (buffer0(2),root_logging_sid_blink)

        integer*4       flink1
        integer*4       blink1
        integer*4       logging_sid
        integer*4       root_memory_node_flink
        integer*4       root_memory_node_blink
        integer*4       memory_node_entry_count

        equivalence     (buffer1(1),flink1)
        equivalence     (buffer1(2),blink1)
        equivalence     (buffer1(3),logging_sid)
        equivalence     (buffer1(4),root_memory_node_flink)
        equivalence     (buffer1(5),root_memory_node_blink)
        equivalence     (buffer1(6),memory_node_entry_count)

        integer*4       flink2
        integer*4       blink2
        integer*4       memory_node
        integer*4       root_array_flink
        integer*4       root_array_blink
        integer*4       array_entry_count

        equivalence     (buffer2(1),flink2)
        equivalence     (buffer2(2),blink2)
        equivalence     (buffer2(3),memory_node)
        equivalence     (buffer2(4),root_array_flink)
        equivalence     (buffer2(5),root_array_blink)
        equivalence     (buffer2(6),array_entry_count)

        integer*4       flink3
        integer*4       blink3
        integer*4       array
        integer*4       root_array_bank_flink
        integer*4       root_array_bank_blink
        integer*4       array_bank_entry_count

        equivalence     (buffer3(1),flink3)
        equivalence     (buffer3(2),blink3)
        equivalence     (buffer3(3),array)
        equivalence     (buffer3(4),root_array_bank_flink)
        equivalence     (buffer3(5),root_array_bank_blink)
        equivalence     (buffer3(6),array_bank_entry_count)

        integer*4       flink4
        integer*4       blink4
        integer*4       array_bank
        integer*4       root_array_bit_flink
        integer*4       root_array_bit_blink
        integer*4       array_bit_entry_count

        equivalence     (buffer4(1),flink4)
        equivalence     (buffer4(2),blink4)
        equivalence     (buffer4(3),array_bank)
        equivalence     (buffer4(4),root_array_bit_flink)
        equivalence     (buffer4(5),root_array_bit_blink)
        equivalence     (buffer4(6),array_bit_entry_count)

        integer*4       flink5
        integer*4       blink5
        integer*4       array_bit
        integer*4       error_count

        equivalence     (buffer5(1),flink5)
        equivalence     (buffer5(2),blink5)
        equivalence     (buffer5(3),array_bit)
        equivalence     (buffer5(4),error_count)

        integer*4       insert_blink
        integer*4       logging_sid_entry_count
        integer*4       logging_sid_entry_address
        integer*4       memory_node_entry_address
        integer*4       array_entry_address
        integer*4       array_bank_entry_address
        integer*4       array_bit_entry_address
        integer*4       search_sid
        integer*4       search_memory_node
        integer*4       search_array
        integer*4       search_array_bank
        integer*4       search_array_bit
        Integer*4       I
        Integer*4       J
        Integer*4       K
        Integer*4       L
        Integer*4       M
        Integer*4       lib$extzv
        integer*4       compress4

        byte            lun

C
C Get the root flink and neccessary entry counts fort the memory information
C queue.
```

```
C
        call GET_MEMORY_Q_INFO (root_logging_sid_flink,logging_sid_entry_count,
        1 memory_node_entry_count,array_entry_count,array_bank_entry_count,
        2 array_bit_entry_count)

        logging_sid_entry_address = root_logging_sid_flink

        do 200,i = 1,logging_sid_entry_count

        call movc3 (%val(24),%val(logging_sid_entry_address),buffer1)

        call frctof (lun)

        call linchk (lun,2)

        write(lun,110) logging_sid
 110    format(/' ','SUMMARY OF MEMORY ERRORS LOGGED BY SID ',z8.8)

        memory_node_entry_address = root_memory_node_flink

        do 190,j = 1,memory_node_entry_count

        call movc3 (%val(24),%val(memory_node_entry_address),buffer2)

        call linchk (lun,5)

        if (lib$extzv(24,8,logging_sid) .eq. 1) then

        write(lun,115) 'TR #',memory_node
 115    format(/' ','CONTROLLER AT ',a,i<compress4 (memory_node)>,'.')

        else if (lib$extzv(24,8,logging_sid) .eq. 2) then

        write(lun,115) 'SLOT INDEX #',memory_node
        endif

        write(lun,117) 'ARRAY#','BIT#','BANK','CORRECTED','FATAL',
        1 'ERRORS','ERRORS'
 117    format(/' ',t8,a,t16,a,t24,a,t35,a,t50,a,/,
        1 t37,a,t50,a)

        array_entry_address = root_array_flink

        do 180,k = 1,array_entry_count

        call movc3 (%val(24),%val(array_entry_address),buffer3)

        array_bank_entry_address = root_array_bank_flink

        do 170,l = 1,array_bank_entry_count

        call movc3 (%val(24),%val(array_bank_entry_address),buffer4)

        array_bit_entry_address = root_array_bit_flink

        do 160,m = 1,array_bit_entry_count
```

```
                call movc3 (%val(16),%val(array_bit_entry_address),buffer5)

                call linchk (lun,2)

                if (
                1 array .ne. -1
                1 .and.
                1 array_bank .ne. -1
                1 .and.
                1 array_bit .ne. -1
                1 ) then

                write(lun,120) array,array_bit,array_bank,error_count
        120     format(/' ',t10,i2.2,'.',t17,i2.2,'.',t25,i2.2,'.',t31,i10.1,'.')

                else if (
                1 array .ne. -1
                1 .and.
                1 array_bank .ne. -1
                1 .and.
                1 array_bit .eq. -1
                1 ) then

                write(lun,125) array,array_bank,error_count
        125     format(/' ',t10,i2.2,'.',t25,i2.2,'.',t44,i10.1,'.')

                else if (
                1 array .NE. -1
                1 .AND.
                1 array_bank .EQ. -1
                1 .AND.
                1 array_bit .EQ. -1
                1 ) then

                Write (lun,127) array,error_count
        127     Format (' ',T10,I2.2,'.',T17,'N/A',T25,'N/A',T44,I10.1,'.')
                Else

                write(lun,130) error_count
        130     format(/' ',t44,i10.1,'.')
                endif

        155     array_bit_entry_address = flink5

        160     continue

        165     array_bank_entry_address = flink4

        170     continue

        175     array_entry_address = flink3

        180     continue

        185     memory_node_entry_address = flink2

        190     continue
```

```

     195      logging_sid_entry_address = flink1

     200      continue

              return

              end
```

```
PROGRAM SECTIONS

    Name                                Bytes   Attributes

  0 $CODE                                 916   PIC CON REL LCL   SHR   EXE    RD NOWRT LONG
  1 $PDATA                                286   PIC CON REL LCL   SHR NOEXE    RD NOWRT LONG
  2 $LOCAL                                424   PIC CON REL LCL NOSHR NOEXE    RD   WRT LONG

    Total Space Allocated                1626

ENTRY POINTS

    Address   Type  Name

  0-00000000        MEMORY_DISPLAY

VARIABLES

    Address   Type  Name                                       Address   Type  Name

  2-00000030  I*4   ARRAY                                    2-00000018  I*4   ARRAY_BANK
  2-0000008C  I*4   ARRAY_BANK_ENTRY_ADDRESS                 2-0000003C  I*4   ARRAY_BANK_ENTRY_COUNT
  2-00000008  I*4   ARRAY_BIT                                2-00000090  I*4   ARRAY_BIT_ENTRY_ADDRESS
  2-00000024  I*4   ARRAY_BIT_ENTRY_COUNT                    2-00000088  I*4   ARRAY_ENTRY_ADDRESS
  2-00000054  I*4   ARRAY_ENTRY_COUNT                        2-0000005C  I*4   BLINK1
  2-00000044  I*4   BLINK2                                   2-0000002C  I*4   BLINK3
  2-00000014  I*4   BLINK4                                   2-00000004  I*4   BLINK5
  2-0000000C  I*4   ERROR_COUNT                              2-00000058  I*4   FLINK1
  2-00000040  I*4   FLINK2                                   2-00000028  I*4   FLINK3
  2-00000010  I*4   FLINK4                                   2-00000000  I*4   FLINK5
  2-000000A8  I*4   I                                        2-00000078  I*4   INSERT_BLINK
  2-000000AC  I*4   J                                        2-000000B0  I*4   K
  2-000000B4  I*4   L                                        2-00000060  I*4   LOGGING_SID
  2-00000080  I*4   LOGGING_SID_ENTRY_ADDRESS                2-0000007C  I*4   LOGGING_SID_ENTRY_COUNT
 AP-00000004@ L*1   LUN                                      2-000000B8  I*4   M
  2-00000048  I*4   MEMORY_NODE                              2-00000084  I*4   MEMORY_NODE_ENTRY_ADDRESS
  2-0000006C  I*4   MEMORY_NODE_ENTRY_COUNT                  2-00000038  I*4   ROOT_ARRAY_BANK_BLINK
  2-00000034  I*4   ROOT_ARRAY_BANK_FLINK                    2-00000020  I*4   ROOT_ARRAY_BIT_BLINK
  2-0000001C  I*4   ROOT_ARRAY_BIT_FLINK                     2-00000050  I*4   ROOT_ARRAY_BLINK
  2-0000004C  I*4   ROOT_ARRAY_FLINK                         2-00000074  I*4   ROOT_LOGGING_SID_BLINK
  2-00000070  I*4   ROOT_LOGGING_SID_FLINK                   2-00000068  I*4   ROOT_MEMORY_NODE_BLINK
  2-00000064  I*4   ROOT_MEMORY_NODE_FLINK                   2-0000009C  I*4   SEARCH_ARRAY
  2-000000A0  I*4   SEARCH_ARRAY_BANK                        2-000000A4  I*4   SEARCH_ARRAY_BIT
  2-00000098  I*4   SEARCH_MEMORY_NODE                  2-00000094  I*4   SEARCH_SID

ARRAYS

    Address   Type  Name                    Bytes  Dimensions

  2-00000070  I*4   BUFFER0                     8  (2)
  2-00000058  I*4   BUFFER1                    24  (6)
  2-00000040  I*4   BUFFER2                    24  (6)
  2-00000028  I*4   BUFFER3                    24  (6)
  2-00000010  I*4   BUFFER4                    24  (6)
  2-00000000  I*4   BUFFER5                    16  (4)

LABELS

    Address   Label       Address   Label       Address   Label       Address   Label       Address   Label       Address   Label

  1-00000042  110'     1-00000073  115'     1-00000092  117'     1-000000AD  120'     1-000000D2  125'     1-000000EF  127'
  1-00000111  130'             **  155              **  160              **  165              **  170              **  175
          **  180              **  185              **  190              **  195              **  200

FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name                              Type  Name                              Type  Name

   I*4  COMPRESS4                               FRCTOF                                  GET_MEMORY_Q_INFO
   I*4  LIB$EXTZV                               LINCHK                                  MOVC3

COMMAND QUALIFIERS

  FORTRAN /LIS=LIS$:MEMORYS/OBJ=OBJ$:MEMORYS MSRC$:MEMORYS

  /CHECK=(NOBOUNDS,OVERFLOW,NOUNDERFLOW)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /STANDARD=(NOSYNTAX,NOSOURCE_FORM)
  /SHOW=(NOPREPROCESSOR,NOINCLUDE,MAP)
  /F77  /NOG_FLOATING  /I4  /OPTIMIZE  /WARNINGS  /NOD_LINES  /NOCROSS_REFERENCE  /NOMACHINE_CODE  /CONTINUATIONS=19

COMPILATION STATISTICS

  Run Time:            34.31 seconds
  Elapsed Time:        68.12 seconds
  Page Faults:         470
  Dynamic Memory:      236 pages
```

0151 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

MESSAGE
LIS

ML11
LIS

MSCP
LIS

MFTAPE
LIS

MOUNT
LIS

MEMORYS
LIS

MOUXX
LIS

MCHK_DISP
LIS